5-2016

Multilevel Methods for Sparsification and Linear Arrangement Problems on Networks

Emmanuel John
Clemson University, emmanuel.u.john@gmail.com

MULTILEVEL METHODS FOR SPARSIFICATION AND LINEAR ARRANGEMENT PROBLEMS ON NETWORKS

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Science

---

by
Emmanuel John
May 2016

Accepted by:
Dr. Ilya Safro, Committee Chair
Dr. Amy Apon
Dr. Brian Dean
Dr. Hongxin Hu

Abstract

The computation of network properties such as diameter, centrality indices, and paths on networks may become a major bottleneck in the analysis of network if the network is large. Scalable approximation algorithms, heuristics and structure preserving network sparsification methods play an important role in modern network analysis. In the first part of this thesis, we develop a robust network sparsification method that enables filtering of either, so called, long- and short-range edges or both. Edges are first ranked by their algebraic distances and then sampled. Furthermore, we also combine this method with a multilevel framework to provide a multilevel sparsification framework that can control the sparsification process at different coarse-grained resolutions. Experimental results demonstrate an effectiveness of the proposed methods without significant loss in a quality of computed network properties.

In the second part of the thesis, we introduce asymmetric coarsening schemes for multilevel algorithms developed for linear arrangement problems. Effectiveness of the set of coarse variables, and the corresponding interpolation matrix is the central problem in any multigrid algorithm. We are pushing the boundaries of fast maximum weighted matching algorithms for coarsening schemes on graphs by introducing novel ideas for asymmetric coupling between coarse and fine variables of the problem.

Dedication

To all African kids still striving to get a good education regardless of the challenges. To my father for leaving me a legacy through education.

Acknowledgments

My special thanks goes to my adviser Dr. Ilya Safro for helping to make this work possible and for helping me believe that I can do it. I want to say thank you to my parents for their unending love and support. I am also grateful to all my friends and family for all the care and support. My friends Kevin Jett, Mahi, and Mihidum encouraged me to go to grad school and even provided some financial support. A quick thank you to my lab colleagues Hayato and Ehsan - thank you for answering all my questions and for all your help. Many people along the way have contributed positively to my life. I cannot mention all the names but I will be forever grateful. Finally, I would like to thank all my committee members for their support and help in my research.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many real world objects and the relationship between them can be modeled as networks which are represented as graphs in the computer systems and algorithms, where vertices represent the objects and edges abstract away the connection between them. For the purpose of this thesis, this definition of "real world" networks would suffice. Problems in scheduling, transportation, VLSI and robotics benefit from being modeled as graph problems. Therefore, the capability afforded by graphs lead to algorithmic problems like searching, spanning trees, shortest path, flow problems and matching problems. Efficient implementations for some of these algorithms exists. However, many graph problems are computationally expensive (e.g., they can be NP-complete). Examples of such problems include graph partitioning, linear arrangement, and coloring. In particular, many versions of, so called, constrained cut-based problems such as partitioning, and clustering are computationally intractable. Quadratic, cubic (and even many nearly-linear time) approximation algorithms that produce suboptimal solutions with some guarantees become infeasible as the size of the input becomes large, so heuristics that provide approximate solutions in strictly linear time with no hidden coefficients are critical. Some of these problems benefit from the application of multiscale methods.

In this thesis, we focus on developing scalable algorithms for real world networks. First we introduce a method of graph sparsification that combines the algebraic distance of edges with a multilevel framework to filter the graph at various resolutions. We provide empirical results that show the preservation of many important structural properties of the original network. Secondly, we develop the minimum 2-sum solver introduced in [66] and extend the coarsening scheme with a stable matching algorithm to improve the solution for real world networks.

Thesis Structure

We begin this thesis with an overview of various methods applied in the domain. In Chapter 1.1, we provide a quick survey and background of multiscale methods. In Chapters 1.2-1.4, we provide an overview of stable matching, graph layout problems and sparsification. Chapter 2 of this thesis is based on an accepted journal paper [41]. The main focus of that chapter is graph sparsification. In Chapter 3, we develop an multilevel algorithm for the minimum 2-sum problem first introduced in [66] and extend the coarsening scheme with a stable matching algorithm.

1.1 Background on Multiscale methods

Data derived from complex networked systems in a form of weighted graphs can exhibit a discrepancy between the macro- and the microscopic scales. This is due to the difference in the underlying physical, biological or social models that describe the system at different scales. In many cases, it has been observed that complex and even non-deterministic systems can exhibit a much more ordered behavior at their coarse-grained resolutions. Multiscale methods are a class of algorithms that are employed in large scale computational and optimization problems to efficiently produce good approximate solutions in which the information from different scales of coarseness is used. Depending on the application and the domain, these methods are also referred to as multilevel, multiresolutional, and multigrid techniques. In this thesis we use them interchangeably.

Multiscale methods are typically applied to large-scale problems in which we can expect to interpolate a solution of one variable to another. In other words, we can find dependencies between variables. An example is in the solving of partial differential equations where the change in the error at each iteration is small. Solving the entire problem in one shot can be prohibitively expensive. However, solving such problems at varying scales using multilevel algorithms, in which a solution in one point can be used to approximate another point, help to improve the solution and eliminate the waste to computational resources [9]

Our multilevel algorithms for graphs are inspired by the algebraic multigrid [14]. In graphs, the multilevel techniques work by aggregating parts or full vertices of the graph recursively to form a successive hierarchy of coarse graphs, G_0, G_1 ,..., G_k in increasing coarseness where G_0 is the finest (original) graph and G_k is the coarsest graph. When the hierarchy of coarse representations is constructed, the computational problem is solved for the coarsest level and the solution is then refined and propagated to all the graphs staring from the coarsest to the finest. (See Figure 1.1). The idea is based on the intuition that vertices that have some

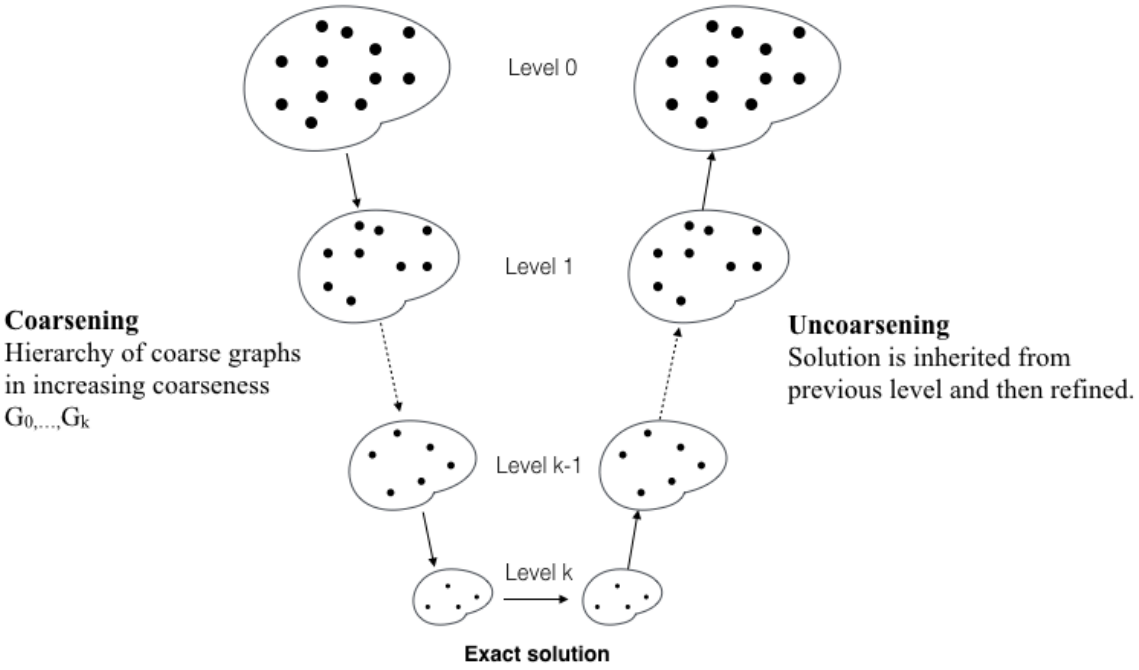


Figure 1.1: Coarsening and Uncoarsening cycle of a multilevel algorithm

"similarity, and closeness" properties might share similar solutions. For example, in the graph partitioning they may belong to the same part. This framework tends to accelerate global optimization algorithms and improve the quality of the solution.

The multiscale methods were first developed by Fedorenko and Bakhvalvov [4, 26, 27] and were employed in the solving of elliptic partial differential equations (PDE). The method was however extended by Brandt in his multi-level adaptive technique (MLAT) for solving boundary value problems [8]. Given their scalable nature, multiscale methods have since been applied to different computational or optimization problems in many fields. Applications of the method in previous studies have also been shown to significantly reduce the running times. In addition, it has been shown that local algorithms at each scale can be easily parallelized, making multiscale methods ideal candidates for parallel computers [11]. There is a long history of applying multilevel methods on graphs. Walshaw applied it to the drawing of large graphs [87]. Safro et al. applied the framework for solving the minimum linear arrangement problem in [65] to obtain approximate solutions in computationally efficient time frames. Examples include solvers for linear ordering problems [68] and network compression [70]. Numerous applications exist also for graph partitioning [34], modularity clustering [57], and graph visualization [84] Such problems as the traveling salesman, community detection

and graph layout benefit from a multilevel framework [9, 25, 68, 85]. The multilevel methods are also broadly applied in machine learning making different algorithms applicable on large-scale data. Examples include support vector machines [60], clustering [47], dimensionality reduction [72], and image segmentation [76]. An in-depth theoretical treatment of multilevel algorithms is beyond the scope of this thesis. For a deeper study, more literature on the topic can be found in [14, 15, 55]

Chapter 2

Single- and Multi-level Network Sparsification by Algebraic Distance

2.1 Introduction

Networks are an abstract model of the relationships between discrete objects. Examples include networks of genes, consumers and generators in the power grid, and networks of friendships or followers in social communities. In order to study real world networks, they are often represented as graphs, where the vertices represent the objects and edges model the relationship or interaction between them. Modeling networks this way facilitates the analysis and understanding of many different structural properties of the underlying complex system. Several powerful software packages such as SNAP [50], Pajek[23], NetworkIt [80], NetworkX [31], and Gephi [6] have been developed to provide this capability. However, many complex networks are massive in size. For example, Facebook users post about 3.2 billion likes and comments each day [1], Twitter has more than 190 million users and about 65 million tweets are posted each day [88], and the human gene network contain several million edges [62]. Although, modeling and understanding these networks is very important in many application domains, the massive size of the network makes it often impractical to perform network analysis on the entire dataset.

In sparsification methods, we aim to select a representative sample of the corresponding graph such that some properties of the original graph are preserved. In other words, central to sparsification is the idea that if an algorithm depends on or computes the properties that are preserved in the sparsified graph,

we can expect that the results will be similar for the original graph [36] while the algorithm will perform much faster on the sparsified graph. Sampling is broadly being carried out in real world networks. Most network analytics consider just a sample in time of the networks under study which is usually the result of data collection limitations [1]. Thus, it is important to understand and develop scalable methods for sampling massive networks.

There are several motivating examples for network sparsification. One obvious example is in the domain of visualization. It is often computationally intensive to render huge graphs on a computer screen as well it is hard to visually analyze such graphs. Sparsification helps to visualize a sample of the graph that reveals structural properties that would have been difficult to visualize and visually analyze in the original graph [46, 74]. The computational difficulty of visualization often arises from its objective, which requires solving a computational optimization problem [2, 38]. Another broad application is the reduction in the cost of computational network analysis. In computing the betweenness centrality of every node in a massive network, for example, by prioritizing what edges should be retained and what should be removed, it is possible to improve the running time of the algorithms at a very minimal cost in optimality [3]. Thirdly, graph sparsification can be applied to revealing hidden populations which are hard for researchers to find by just looking at the entire population. For example, Salganik et. al showed that when trying to sample the population of injection drug dealers, it is difficult to sample directly as this population is hidden and so specialized sampling algorithms are needed [73]. Methods applied usually involve starting out with a sample of the desired population and using that as a seed for revealing the other members of the sample population [36]. Existing methods include snowball sampling [29, 32] and respondent driven sampling [73]. In addition, in the case where there is an incomplete data, sampling can be used to estimate properties of the original graph. This is particularly useful in dynamic graphs [82], graph streaming algorithms [1] and collective classification [71].

There are several approaches to sampling a large graph while preserving the desired properties. An example involves formulating a mathematical programming problem to minimize the distance between the sparse graph and the original graph [36]. However, such approaches are often quite complex and running them might be costlier than running the algorithm on the larger graph. Spectral approximation algorithms also exist [79]. However, those algorithms are not very fast as well and often infeasible for large graphs [36] as they often involve hidden constants and require convergence in eigen-problems. The more common approaches are (1) vertex sampling, which involves selecting a number of vertices from the original graph and retaining the vertex-induced subgraph, and (2) edge sampling, which involves the selection of edges and corresponding edge-induced subgraph. Other variations of edge and vertex sampling have been developed

(see [36] for a full survey). In our method we focus on the edge sampling and also preserve the nodes from the original graph. In order to achieve this, we ensure that every node has at least one incident edge in the sparsified graph.

2.1.1 Strength of Connectivity in Sparsification

If the properties to be preserved are known beforehand, then, in many cases, it is possible to determine what kind of edges are important to preserve those properties and which ones are redundant. Thus, the sampling transformation can then be designed with the objective of retaining those edges. A general framework for sparsification involves: (1) ranking the edges and assigning each edge an edge score; and (2) sampling edges based on their scores [36]. Scoring edges provides a motivation for rating the strength of connection between two vertices. In particular, this is extremely important in weighted networks, where the weights can be approximate, noisy or even completely missing. Different types of the connection strength have been proposed for scoring edges. We refer the reader to [51] for a brief survey on the sparsification-relevant types of connectivity strength. The most relevant to our work is a cohort of spectral methods widely used in theoretical computer science to sparsify dense graphs such that some spectral properties are preserved. These are usually cut-based properties that are formulated using Cheeger inequality. For example, Spielman et al. introduced the edge effective resistance [77]. The effective resistance is computed using the linear system solver [78] which runs in $O(mlog^{15}n)$ time which can be time consuming to be feasible. Another example is the vectorized PageRank [22]. Various interpretations of the diffusion have been proposed and analyzed [43, 83] for graph kernels. However, these methods usually suffer from impractical complexity.

Another relevant class of methods is based on the Jaccard index in which a similarity between two vertices is measured by computing the overlap in their neighborhoods. In [74], Satuluri et. al rated edges according to the local similarity $\text{sim}(i,j) = |N_i \cap N_j|/|N_i \cup N_j|$, where N_i is the neighborhood of node i. This method was designed for clustering objectives assuming that nodes with larger shared neighborhoods are likely to belong to the same cluster. A global similarity threshold is then chosen for which edges are filtered. The authors also introduced a method for local sparsification in which they rate and filter edges per node by selecting the top d_i^e edges ranked by their similarity score, where $e \in (0,1)$. Their method ensures that there is at least one edge per node after sparsification. We explore this property in our method. This sparsification technique can be computationally expensive since it requires counting the number of triangles an edge is a part of. The authors, however, provided an approximation for computing the similarity. Based on the work of Satuluri et al. [74], local degree method favors the retention of high degree nodes - also known

as hub nodes [51]. As in the local similarity, for each node, they include edges to the top d_i^e nodes. However, edges are sorted according to the degree of their neighbors in descending order. The main idea of this method is to keep edges in sparsified graph that leads to nodes with high degree. Additionally, vertex connectivity can be measured by the betweenness centrality, the shortest path length, the weight of substructures (such as spanning rooted forests, routes, overlapping paths that connect two vertices [18]) and algebraic distance [19] which we will discuss in Section 2.3.

2.1.2 Our Contribution

We introduce two methods for complex network sparsification that distinguish between strong and weak connectivity through neighborhoods of limited distance from the endpoints of edges. In some networks (such as those that include geospatial information), these types of connections can be interpreted as long- and short-range connections while in other (such as social networks) as inner- and outer-community connections. In both methods the sampling is based on the connectivity measured by the algebraic distance between nodes [19]. It generalizes the idea of methods that estimate the Jaccard coefficient for more distant neighborhoods through limited application of lazy random-walks (also known as algebraic distance [19]).

In the first method (the single level approach) we demonstrate multiple settings of filtering local and global connections with the sampling similar to [74]. In the second method we propose a multilevel algorithm that combines the single level approach with the multilevel framework [61] to sparsify graphs at different coarse-grained resolutions. We provide a robust method that can be tuned to preserve different network properties that are important in a variety of applications. The multi- and single level methods can both be used to either preserve the global structure or the local structure. The resulting sparsified networks are compared with the original network using following properties: degree distribution, clustering coefficient, number of connected components[1], diameter, betweenness centrality, PageRank centrality, and modularity [56]. Evaluation of both methods is demonstrated through comparison of the aforementioned network properties with those measured on the original network. Finally, we discuss how our method can be parallelized and show the running time in OpenMP implementation. The proposed methods are implemented and available at [40].

[1] In many existing sparsification methods, the number of connected components is preserved "artificially", i.e., even if the edge is marked for deletion, it is not deleted if it increases the number of connected components. Here we do not restrict our algorithms with such requirement.

2.2 Preliminaries

We denote the graph underlying a given network by $G = (V, E, w)$, where V is a set of vertices, E is a set of edges, and $w : E \to \mathbb{R}_{\geq 0}$ is a weighting function on E that represents the strength of connectivity between two vertices. The graph is undirected, containing no self-loops and multi-edges. For each node $i \in V$ we define its degree by d_i and its neighbors by N_i. The clustering coefficient is a measure of the probability that neighbors of a node are connected to each other [56]. Consequently, it is a measure of the degree to which nodes in a network tend to cluster [88]. The clustering coefficient of a node i is defined as $c_i = \lambda_i/\tau_i$, where λ_i is the number of triangle subgraphs i participates in, and $\tau_i = d_i(d_i - 1)/2$, i.e., the number of triples. The clustering coefficient of a graph G is defined as

$$C_G = \frac{1}{|V'|} \sum_{i \in V'} c_i, \tag{2.1}$$

where $V' = \{i \in V \mid d_i > 1\}$. The diameter of a graph is defined as the maximum distance shortest path among all pairs of vertices in G from the same connected component. The resulting sparsified networks are compared with the original network using following properties: degree distribution, clustering coefficient, number of connected components[2], diameter, betweenness centrality, PageRank centrality, and modularity [56]. We will use the Spearman Rank Correlation Coefficient (ρ) that is a measure of the correlation between two distributions. It is defined as $\rho = 1 - (6 \sum p_i^2)/(n(n^2 - 1))$, where $p_i = x_i - y_i$, and x_i and y_i are the ranks computed from the scores X_i and Y_i.

2.3 Algebraic distance

In order to determine the strength of connection of edges for the purpose of sparsification, we use the algebraic distance introduced in [19, 61]. The algebraic distance of an edge ij (denoted by δ_{ij}) is interpreted as locally converged iterative process that propagates the weighted average of values from N_i and N_j initialized by random numbers [19]. This expresses the strength of connectivity between two nodes through their local neighborhoods. The process is essentially a Jacobi overrelaxation (JOR) or a lazy random walk with limited number of steps (see Algorithm 1). The algebraic distance was successfully used in several algebraic multigrid algorithms [16, 52] and in multilevel algorithms for discrete optimization on graphs (such

[2]In many existing sparsification methods, the number of connected components is preserved "artificially", i.e., even if the edge is marked for deletion, it is not deleted if it increases the number of connected components. Here we do not restrict our algorithms with such requirement.

as the minimum linear arrangement [61], and graph partitioning [69]) to reduce the order of interpolation that results in a sparsified coarse system.

Algorithm 1 Algebraic distance implementation: ComputeAlgDist

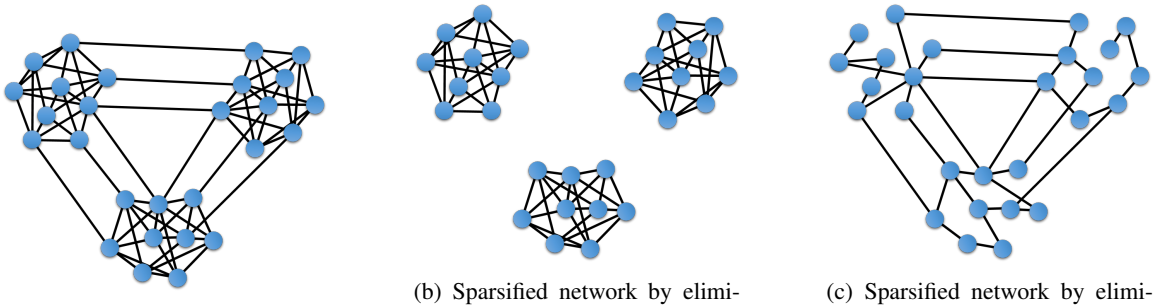
Input: Parameter α (in our experiments $\alpha = 1/2$)
$\forall ij \in E\ R_{ij} = 0$
for $r = 0, 1, 2, ...$ **do** ▷ the number of test vectors r is small
  $\forall i \in V\ x_i^{(0)} \leftarrow rand(-0.5, 0.5)$
  for $k = 0, 1, 2, ...$ **do** ▷ the number of JOR iterations k is small
    $\forall i \in V\ x_i^{(k)} \leftarrow \alpha x_i^{(k-1)} + (1-\alpha)\dfrac{\sum_{j \in N_i} w_{ij} x_j^{(k-1)}}{\sum_{j \in N_i} w_{ij}}$
  end for
  Rescale x back to $(-0.5, 0.5)$
  $\forall ij \in E\ R_{ij} = R_{ij} + (x_i - x_j)^2$
end for
return $\forall ij \in E\ \delta_{ij} \leftarrow \dfrac{1}{\sqrt{R_{ij} + \epsilon}}$ ▷ ϵ is sufficiently small
$\forall ij \in E\ \delta_{ij} \leftarrow \dfrac{\delta_{ij}}{\sqrt{d_i * d_j}}$ ▷ optional normalization

Other stationary iterative relaxations can also be applied in a similar setting but since JOR is implicitly parallelizable using matrix-vector multiplications, we prefer to use it instead of other relaxations (such as Gauss-Seidel) that converge faster. Optionally, the algebraic distance can also be normalized by the square-root of the product of the weighted degrees of the two nodes to reduce extremely high strength of connection between hub nodes.

The algebraic distance will serve as the main criterion for choosing edges for sparsification in the algorithms below. Because it helps to distinguish between so called short- and long-range connections [19], we will use it to demonstrate different types of sparsification in which local and global properties are preserved correspondingly to the types of algebraic distances that we choose. The short-range connections (large values of δ_{ij}) will be called δ-strong. The long-range connections (small values of δ_{ij}) will be called δ-weak.

2.4 Single-level sparsification

In the single-level approach we demonstrate three types of sparsification in which we filter δ-weak, δ-strong edges and their mixture. In all of these cases, first, for each edge in the graph, we compute the algebraic distance. Then, for each node i, we sample the top d_i^e neighbors ranked by their algebraic distances, where $e \in [0, 1]$. In this approach it is possible to sample for local or global structure preservation or a combination of both. To preserve the global structure, we select d_i^e weakest connections and add them to the



(a) Original network

(b) Sparsified network by eliminating δ-weak connections

(c) Sparsified network by eliminating δ-strong connections

Figure 2.1: An example of a small network with 3 dense clusters and sparse cuts between them (a). Sparsification of δ-weak connections will result in network presented in (b). Sparsification of δ-strong connections is presented in (c).

sparse graph (see Figure 2.1c). Similarly, d_i^e strongest connections are preserved to emphasize the importance of a local structure in the sparse graphs (see Figure 2.1b).

Algorithm 2 Single-level sparsification: Sparsify(G)

1: **Input:** Sparsification parameter e, Graph G

2: **Output:** Sparsified graph G_{sparse}

3: $G_{sparse} \leftarrow$ empty graph

4: COMPUTEALGEBRAICDISTANCES(G)

5: **for** $i \in V$ **do**

6: Sort N_i by δ_{ij} in ascending (or descending) order

7: Add top d_i^e edges to G_{sparse}

8: **end for**

9: **return** G_{sparse}

It is also possible to partially preserve both global and local structures with a slight change in the algorithm, namely, by distributing the algebraic distances into bins, and sampling the edges from all bins. In order, to distribute the algebraic distances into bins, we define the bin width $h = \frac{3.5*\sigma}{\sqrt[3]{d_i}}$, and the number of bins $k = (\max \delta_{ij} - \min \delta_{ij})/h$, where σ is the standard deviation of algebraic distances.

2.5 Multilevel sparsification

The multilevel approach [14, 86] can be applied as a general framework for many different numerical methods. Most real-world instances are not completely random, i.e., a particular similarity or dependence

Algorithm 3 Single-level sparsification with binning: SparsifyB(G)

Input: Sparsification parameter e, Graph G
Output: Sparsified graph G_{sparse}
$G_{sparse} \leftarrow$ empty graph
COMPUTEALGEBRAICDISTANCES(G)
for $i \in V$ **do**
    Distribute N_i into bins, each bin corresponds to edges ...
    Randomly select bins and edges up to d_i^e to G_{sparse}
end for
return G_{sparse}

between variables exists and, thus, can partially be detected to reduce their number in complex computations. Here we introduce and advocate the use of multilevel approach as a general purpose framework for network sparsification. In the heart of the proposed method lies an idea to sparsify the network at multiple scales of coarseness which, in contrast to most existing sparsification methods that sample single edges, will allow to sample clusters of edges of different sizes and δ-weakness.

It is known that the topology of many complex networks is hierarchical (or multiscale) and, thus, often might be self-dissimilar across scales [7, 39, 54, 89]. In such hierarchical representations, groups of nodes are aggregated into communities, which automatically bundles edges into coarse connections. Bundling the edges at different scales of coarseness will introduce different levels of δ-weakness for such coarse connections which may or may not be required to be sparsified for the required analysis. For example, in the analysis of a social network, we may want to visualize only a certain type of edges that connect dense communities of small sizes, while connections between large communities and local inner connections are out of the scope. In the proposed framework, this can be achieved by creating a hierarchy of coarse representations, and sparsifying at those levels that do not correspond to the desired communities. To create a multilevel framework we use the algebraic multigrid (AMG) aggregation strategy that was introduced in [65]. For simplicity, we do not split fine nodes across the aggregates (like in some optimization problems [65, 69]) but instead cover the graph with star-like structures and coarsen them. For the completeness of paper we briefly repeat the main components of the coarsening algorithm.

Given an original graph G, in the multilevel framework we recursively construct a hierarchy of decreasing size coarse graphs $G_0 = G, G_1, ..., G_l$. The original graph is gradually coarsened into the smaller graphs until the small enough graph G_l is reached. The sparsification algorithm is then run on the coarsest level and the results (i.e., edges to eliminate for sparsification) are inherited by the finer graph and the uncoarsening continues until G_0 is reached. In most cases, our discussion is focused on fine-to-coarse and

coarse-to-fine transformations of graphs and solutions, respectively. For this purpose, we denote the fine and coarse level graphs by $G_f = (V_f, E_f)$, and $G_c = (V_c, E_c)$, respectively. At each level, after sparsifying edges inherited from G_c, Algorithm 1 is applied to recompute algebraic distance on G_f.

The Coarsening We begin with selecting a dominating set of seed nodes $C \subset V_f$ that will serve as centers of future coarse nodes in V_c. Setting initially $F = V_f$ and $C = \emptyset$, the selection is done by traversal of F and moving to C such nodes that are not strongly coupled to those that are already in C. At each step $F \cup C = V_f$ is preserved, and at the end the size of V_c is known, namely, $|V_c| = |C|$. After C is selected, nodes in $F = V \backslash C$ are distributed to their aggregates according to the restriction operator $P \in \{0,1\}^{|V_f| \times |C|}$, where

$$P_{iJ} = \begin{cases} 1 & \text{if } i \in F,\ J = I_c\left(\operatorname{argmax}_{j \in C} \dfrac{\delta_{ij}}{\sum\limits_{k \in C} \delta_{ik}} \right) \\ 1 & \text{if } i \in C,\ J = I_c(i) \\ 0 & \text{otherwise,} \end{cases} \tag{2.2}$$

and $I_c(j)$ returns an index of coarse node J that corresponds to $j \in C$. Then, the Galerkin coarsening creates a coarse graph Laplacian $L_c = P^T L_f P$, where L_f is the Laplacian of G_f.

Coarsest Level At the coarsest level, we sparsify the edges by using the single-level Algorithm (3). These edges correspond to bundles of edge chains at the fine levels that connect the most distant regions in a graph, so if the goal is to preserve the global structure, the user should avoid of sparsification at deep coarse levels.

Uncoarsening We initialize the solution (sparsification) of G_f by uncoarsening the edges sparsified in G_c. When the order of interpolation in the multilevel algorithm equals 1 (i.e., there is only one non-zero entry per row in P, see Eq. 2.2), each coarse edge $IJ \in E_c$ can bundle at most two types of edges in E_f, namely, at most one edge that connect two seeds $I_c^{-1}(I)$ and $I_c^{-1}(J)$, and possibly multiple edges $pq \in E_f$ such that $P_{pI_c^{-1}(I)} = 1$, and $P_{qI_c^{-1}(J)} = 1$. If IJ is sparsified at the coarse level, then edges of both types are sparsified at the fine level. After initialization of the fine level, we recompute algebraic distances to update the information about connectivity in the sparsified fine graph, and, then, more edges may or may not be sparsified at the fine level depending on the parameter settings. Full multilevel cycle is presented in Algorithm 4. Example of full multilevel cycle on a Facebook network (see fb-uf in Table 2.1) is shown in Figure 2.2.

Algorithm 4 Multilevel sparsification of graph: MLSparsify

Input: fine graph $G_f = (V_f, E_f)$, vector of sparsification ratios
Output: sparse graph $G_f^s = (V_f, E_f^s)$
function ML(G_f)
    COMPUTEALGDIST(G_f)
    if $|V_f|$ is small enough **then**
        $E_f^s \leftarrow$ SPARSIFYB(G_f) ▷ Sparsify coarse edges
    else
        CREATESEEDS(G_f) ▷ Coarsening: seeds
        Compute P ▷ Coarsening: restriction operator
        $G_c \leftarrow (L_c = P^T L_f P)$ ▷ Coarsening: coarse graph
        $G_c^s \leftarrow$ MLSPARSIFY(G_c) ▷ Recursive call to sparsify the next coarser level
        $G_f^s \leftarrow$ UNCOARSEN(G_c^s) ▷ Sparsification of edges inherited from coarse level
        COMPUTEALGDIST(G_f^s) ▷ Algebraic distances are recomputed
        $G_f^s \leftarrow$ SPARSIFYB(G_f^s) ▷ Sparsification of current level edges
    end if
end function
return G_f^s

2.6 Computational Results

Implementation and Evaluation We provide C++ implementation for both the single- and multilevel algorithms in [40]. For the comparison of original and sparsified networks, we employed methods implemented in NetworKit [80]. We experimented with varying degrees of sparsification, taking values of e ranging from 0.1 to 0.9 (see Section 2.4). All numerical properties for the comparison are the averages over 10 runs with different random seeds for each parameter setting. The following parameter were used in computation of algebraic distance: $R = 10$, $k = 40$, and $\alpha = 0.5$. Their robustness is discussed in [19]. In addition, for the single-level algorithm (Algorithm 3), we provide two sets of results for each graph, namely, with and without the normalization (see last step in Algorithm 1) of algebraic distance. In each case we experimented with sparsification of weak edges, strong edges and mixture of both.

In the multilevel algorithm (Algorithm 4), we experimented with sparsifying at the coarsest, middle and the finest levels. In our experiments, we split the number of levels in the multilevel algorithm into 3 equal segments, and choose a parameter, level-span which determine how many levels in each segment gets sparsified. We then sparsify one segment at a time and observe the corresponding network properties. For example, for a graph with 6 levels, with a level-span of 2, to sparsify the coarsest levels only we use the following parameter configuration: $(0.3, 0.3, -1, -1, -1, -1)$, where a setting of -1 indicates no sparsification occurs at this level. Similarly, the middle and finest levels can be sparsified using $(-1, -1, 0.3, 0.3, -1, -1)$, and $(-1, -1, -1, -1, 0.3, 0.3)$ configuration settings respectively. However, in our implementation, users

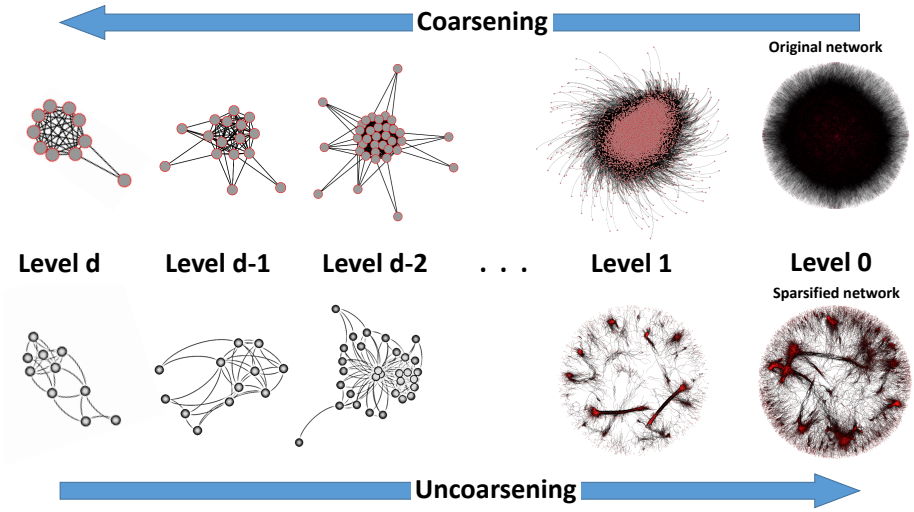


Figure 2.2: Complete Sparsification V-Cycle

can specify any combination of settings for different levels. The sparisification ratio (ratio of number edges in the sparse graph to the number of edges in the original graph), is kept between 20% to 40% for each stage in order to make the results comparable. For the purpose of our study, we maintain a level span of 3.

Datasets We experiment with 18 real-world networks (see Table 2.1), which for the purpose of our study we grouped into two groups of social networks, one group of citation networks (CIT) and one group of biological networks (BIO). We split the social networks into 2 groups (SN1, and SN2) - one consisting of Facebook networks, Livejournal and Google+ (general purpose social networks), and the other consisting other consisting of Flickr, Buzznet, Foursquare, Catster, Blogcatalog and Livemocha. The graphs were retrieved from the NetworkRepository [63], the Koblenz [45], and the SNAP [49] collections. The size of the networks range between 1 million to 34 million edges.

Table 2.1: Benchmark graphs

Group	Graph	$\|V\|$	$\|E\|$	min deg.	max deg.	avg degree
Social	fb-indiana	29.7K	1.3M	1	1.4K	87
Networks 1	fb-texas84	36.4K	1,6M	1	6.3K	87
(SN1)	fb-uf	35.1K	1.5M	1	8.2K	83
	fb-penn94	41.5K	14M	1	4.4K	65
	livejournal	4M	27.9M	1	2.7K	13
	google-plus	107.6K	12.2M	1	20.1K	227.4
Biological	human-gene1	22K	12M	1	7.9K	1.1K
Networks	human-gene2	14K	9M	1	7.2K	1.3K
(BIO)	Mouse	43K	14.5M	1	8K	670
Social	flickr	105K	2.3M	7	5.4K	43.7
Networks 2	buzznet	101.2K	2.8M	1	64.3K	54
(SN2)	foursquare	639K	3.2M	1	106.2	10
	catster	149.7K	5.4M	1	80.6K	72
	blogcatalog	88.8K	2.1M	1	9.4K	47
	livemocha	104.1K	2.2M	1	3K	42
Citation	ca-cit-Hepth	22.9K	2.6M	1	11.9K	233.38
Networks	cit-patent	3.7M	16.5M	1	793	8.75
(CIT)	codblp	540.5K	15.2M	1	3.3K	56

Methods of Comparison We studied various levels of sparsification while comparing the following properties of the sparse graph G_s, to those in the original graph G_o. The single value properties are: (a) **Diameter** - We measure the ratio of the diameter in G_o to the new diameter in G_s (in plots "orig diameter/diameter"); (b) **Number of connected components** - we measure the ratio of the number of connected components in G_s to that in G_o (in plots "comp/orig comp"); (c) **Modularity** - we measure the ratio of modularity in G_s to that of G_o (in plots "mod/orig mod", Networkit [80] provides an implementation of the Louvain method). Certain network properties are represented better by their distributions over the nodes. In order to accurately compare the distributions, we use the Spearman rank correlation coefficient. This effectively, reveals how different the sparse graph is from the original in the context of these properties where a correlation value of 1 means they are perfectly correlated and correlation value of 0 means no correlation. The following distributions are compared using the Spearman rank: (a) Node **betweeenness** centrality; (b) **PageRank** centrality; (c) **Degree distribution**; and (d) **Clustering coefficient distribution** (c_i). The method changes slightly in comparing node betweenness centrality. Considering that the cost of computing betweenness for large graphs is very

expensive, we make use of an approximate method provided by Networkit. However, to ensure accuracy we compute this 10 times and take average of the positional rankings and then compute the Spearman rank correlation.

2.6.1 Single-level Algorithm

The single-level algorithm was tested with both unnormalized and normalized algebraic distances. The results for unnormalized algebraic distance are presented in Figures 2.3, 2.4, 2.5, and 2.6 for groups SN1, SN2, CIT, and BIO, respectively. (The results for the normalized algebraic distance can be found in section 2.7.) In each figure, 3 columns, and 7 rows of plots are presented. In all 4 figures: (a) each column corresponds to the type of filtering, i.e., to the types of edges that retain after sparsification; (b) each row corresponds to the type of comparison. Each plot contains several colored curves that correspond to the respective graphs (see vertical legend). One point in each curve corresponds to an average of the measured comparison method over 10 runs for the corresponding edge ratio in each. The x- and y-axes correspond to the sparsification ratio and method of comparison, respectively. In the y-axis of betweenness, PageRank, degree, and clustering coefficient distribution centralities, the Spearman rank is denoted by ρ. For example, we examine the behavior of the degrees in social network Google+ in SN1 when δ-strong edges retain after sparsification. In Figure 2.3, we find a row "Degree centrality" (row 3). The results for retaining δ-strong edges are found in the third column. The black curve corresponds to Google+, where each point is an average of 10 runs.
Note: Most curves do not reach a visible zero of the x-axis. This is because the sparsification is interrupted when the number of edges becomes less than the number of nodes.

δ-weak edges Plots labelled as δ-weak (column 1) are results obtained by retaining only weak edges, when δ-weak edges are preferred during sparsification (i.e, δ-strong edges are deleted). In this type of sparsification, we expect that sparsification of the local structure will mostly dominate the sparsification of the global structure. Indeed, we observe that properties (such as the betweenness centrality, diameter, and the number of components) that heavily depend on usually limited number of long-range weak connections are well preserved.

δ-strong Plots labelled as δ-strong (column 3) are results obtained by retaining δ-strong edges and removing δ-weak edges. By preferring δ-strong edges during sparsification, we attempt to preserve properties that

depends on the local structure of the graph. Such properties as clustering coefficient, pagerank and degree centrality survive sparsification better when this method is used. In particular, we can observe that the clustering coefficient (which is in many cases the reason for a strong community structure) is preserved at the level of $\approx 75\%$ in SN1 when 70% of edges are removed (instead of $\approx 40\%$ for δ-weak sparsification). A similar phenomena is observed in BIO. It is interesting to note that in SN2, in comparison to the δ-weak sparsification, the changes in the clustering coefficient are not significant.

Mixture sparsification In plots labelled as mixed, we maintain a balance between the δ-weak and δ-strong types of sparsification by preferring ensuring that both are sparsified. For such properties as the betweenness centrality, PageRank and degree centrality, the results are better for up to 20% sparsification ratio when compared to selecting either weak or strong edges. For such properties as the clustering coefficient, modularity, diameter and connected components, retaining both weak and strong edges provides results that is in between that produced by weak or strong edges sparsification.

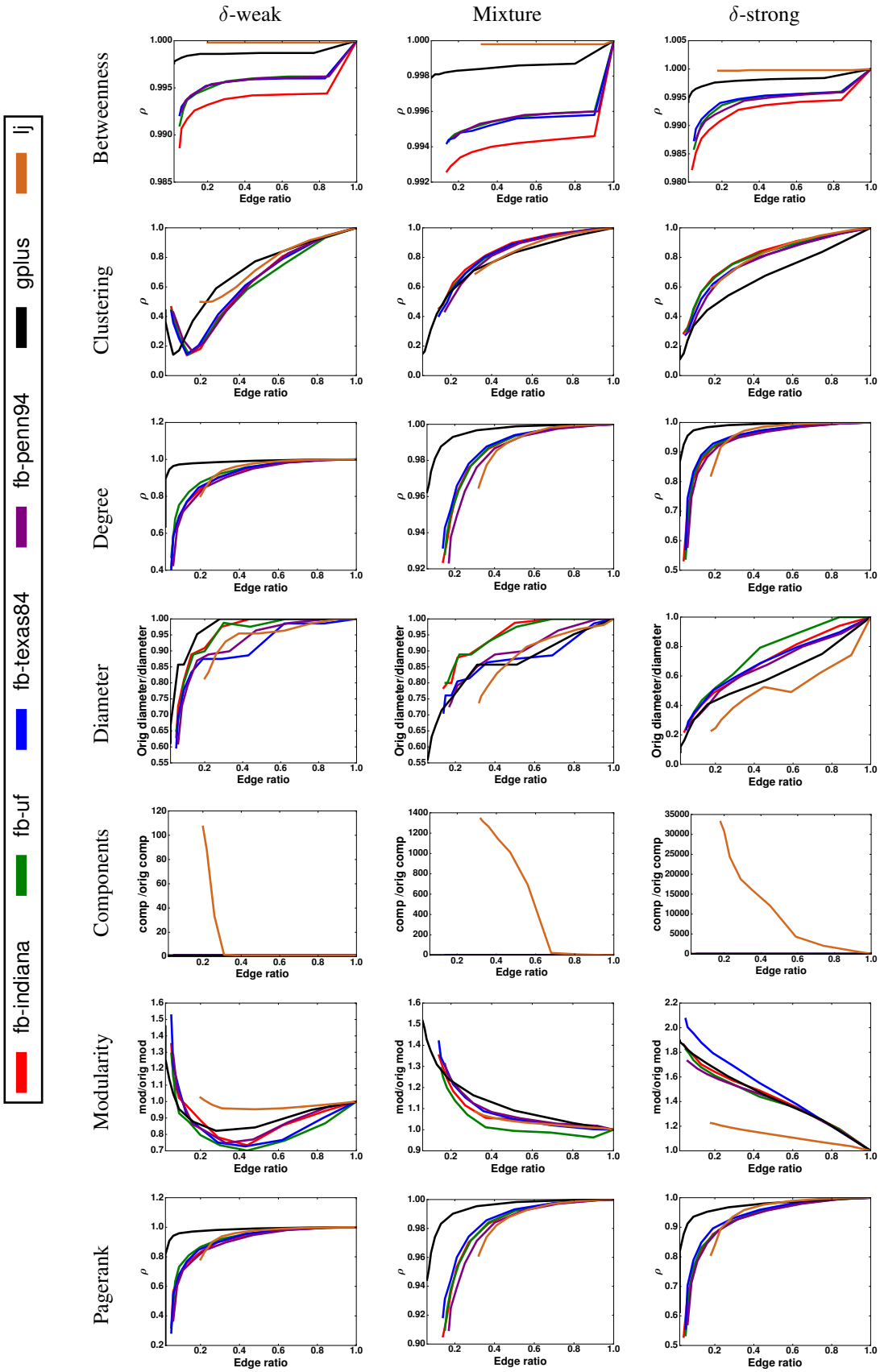


Figure 2.3: Social Networks 1

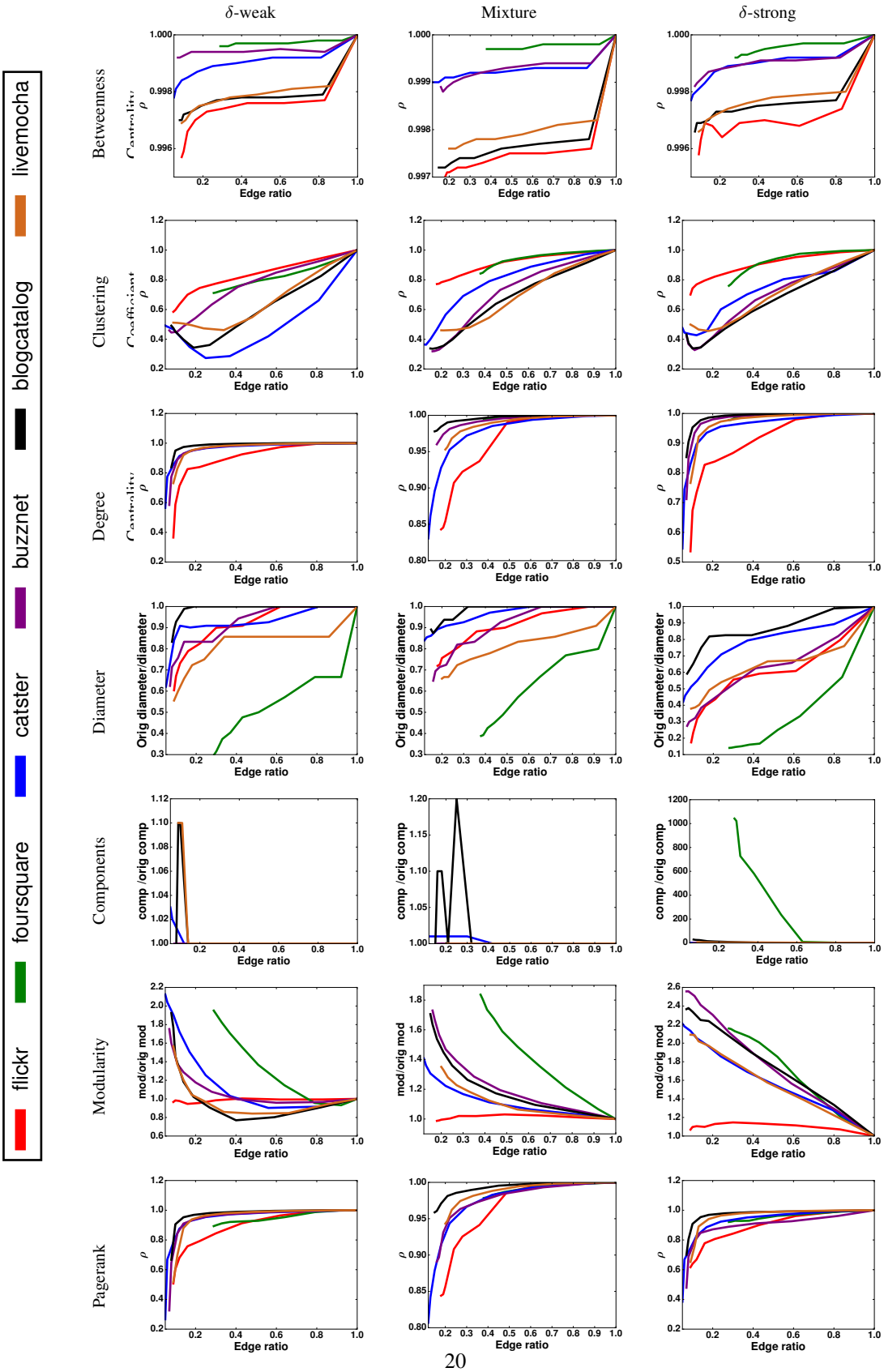


Figure 2.4: Social Networks 2

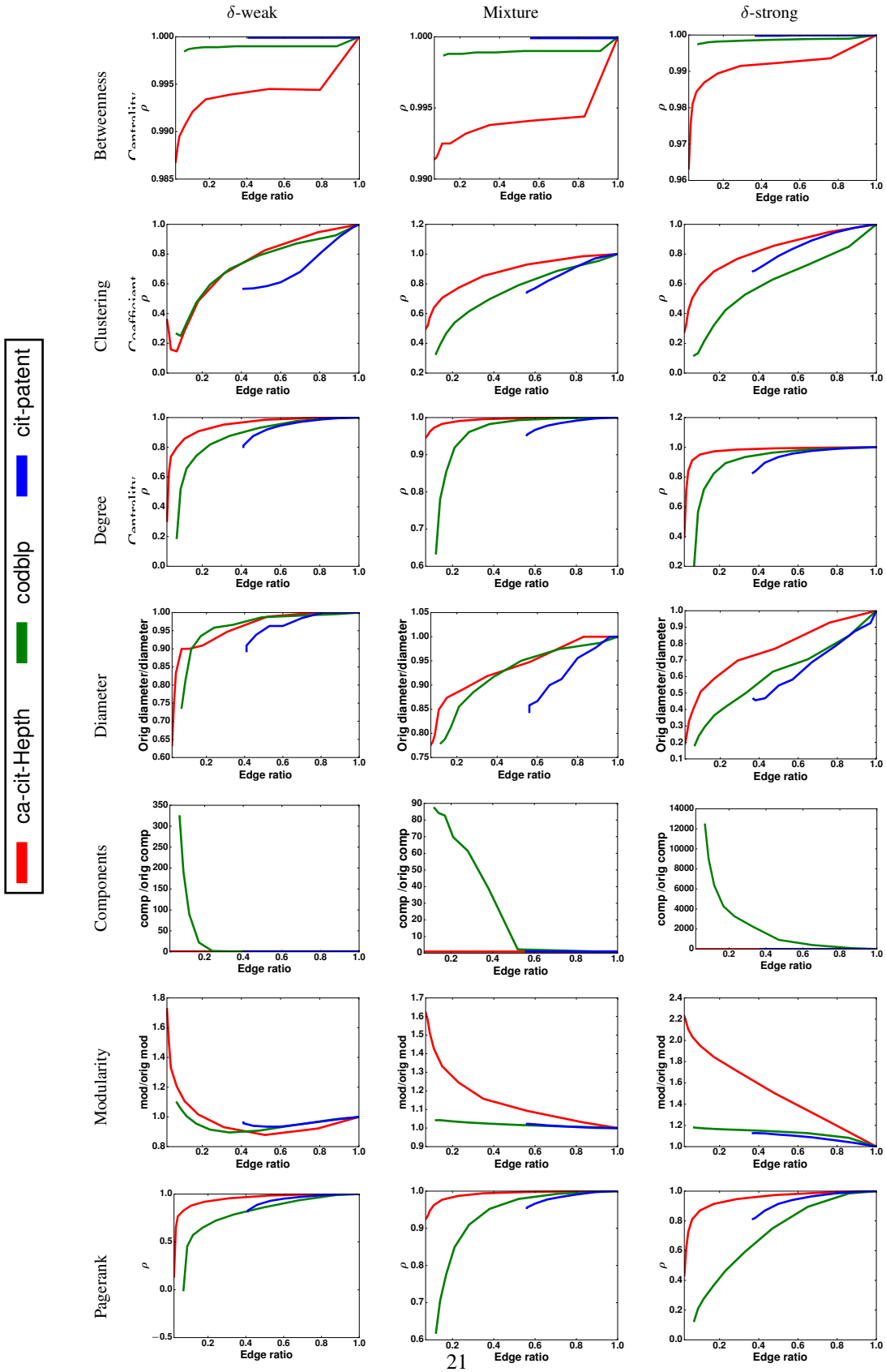


Figure 2.5: Citation Networks

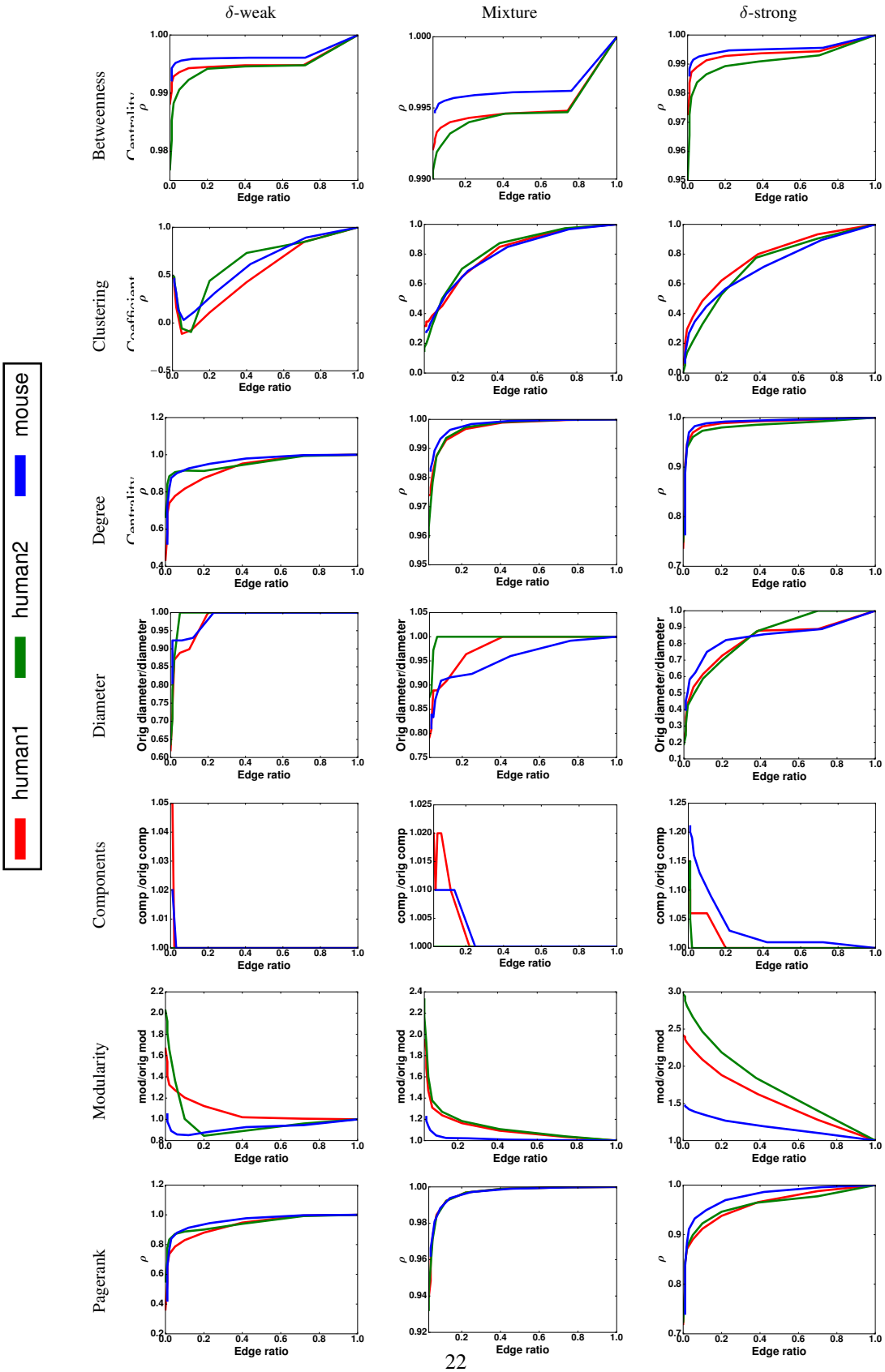


Figure 2.6: Biological Networks

2.7 Normalized Sparsification

We experimented with the single-level algorithm that employ normalized algebraic distances (see line 15 of Algorithm 1). The purpose of this normalization is to decrease the strength of connection expressed in the algebraic distance between hubs. The normalization results show that normalizing the algebraic distance further improves properties that are sensitive to the existence of weak edges. Example are diameter and connected components. As seen in the plots for diameter (see δ-weak column in Figures 2.7, 2.8, 2.9, and 2.10), the minimum edge ratio before the diameter deteriorates is further improved. Similarly for the number of components the number of components for the smallest sparse graph is reduced and some case kept constant as seen in δ-weak column in Figures 2.7, 2.8, 2.9, and 2.10). Such properties as local clustering coefficient, degree centrality, and PageRank that do not depend on global edges are relatively unaffected.

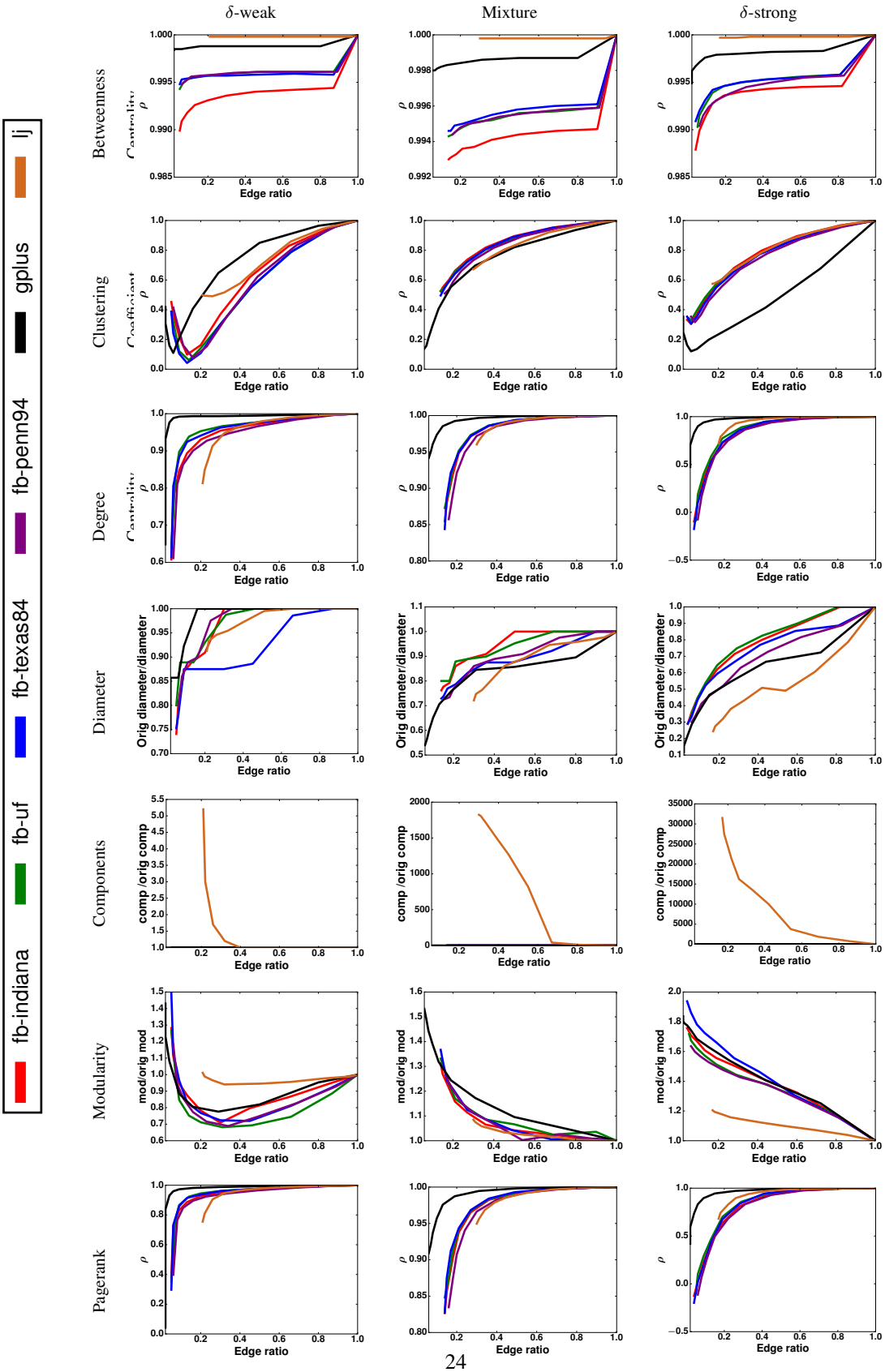


Figure 2.7: Social Networks 1

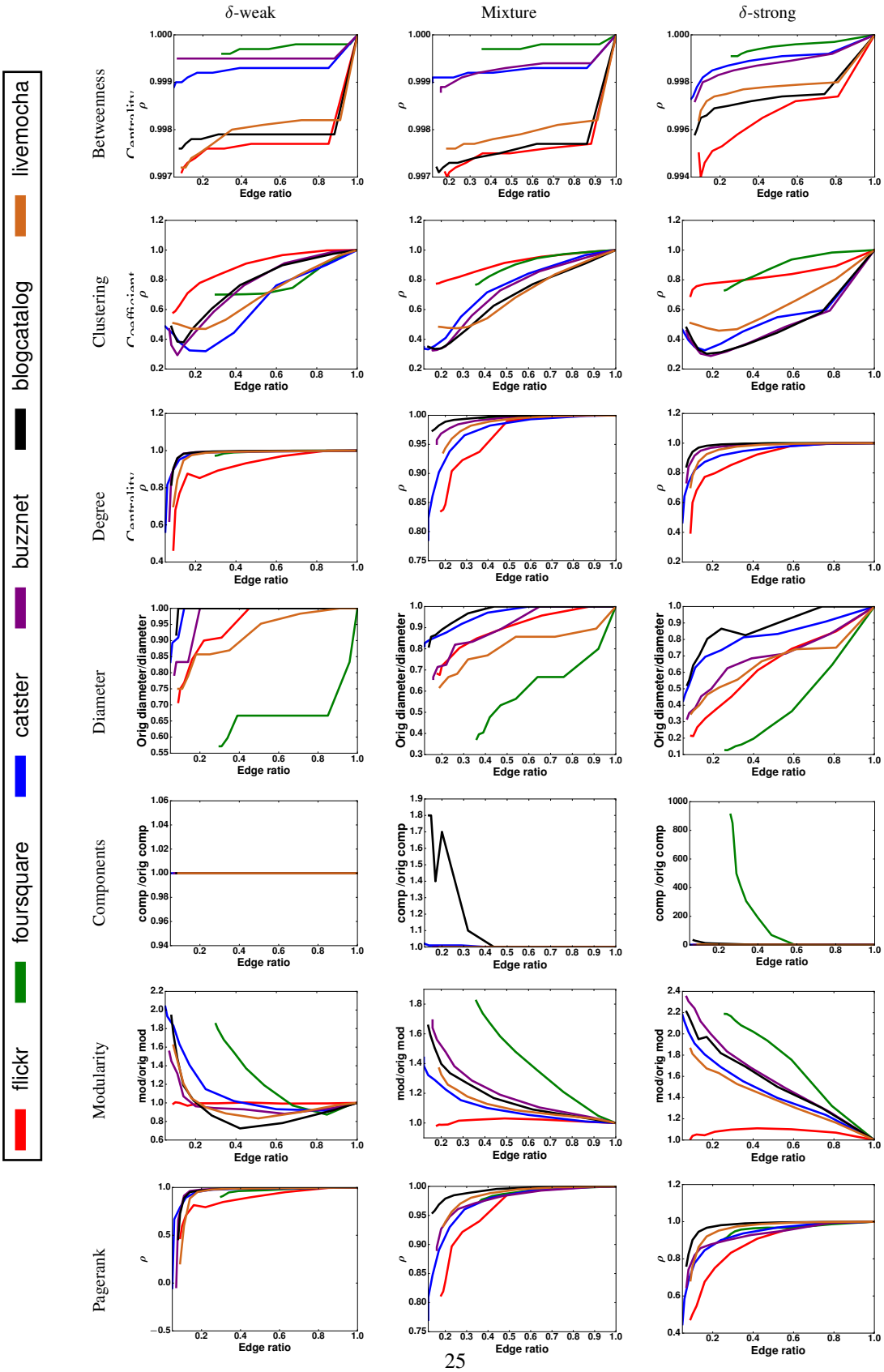


Figure 2.8: Social Networks 2

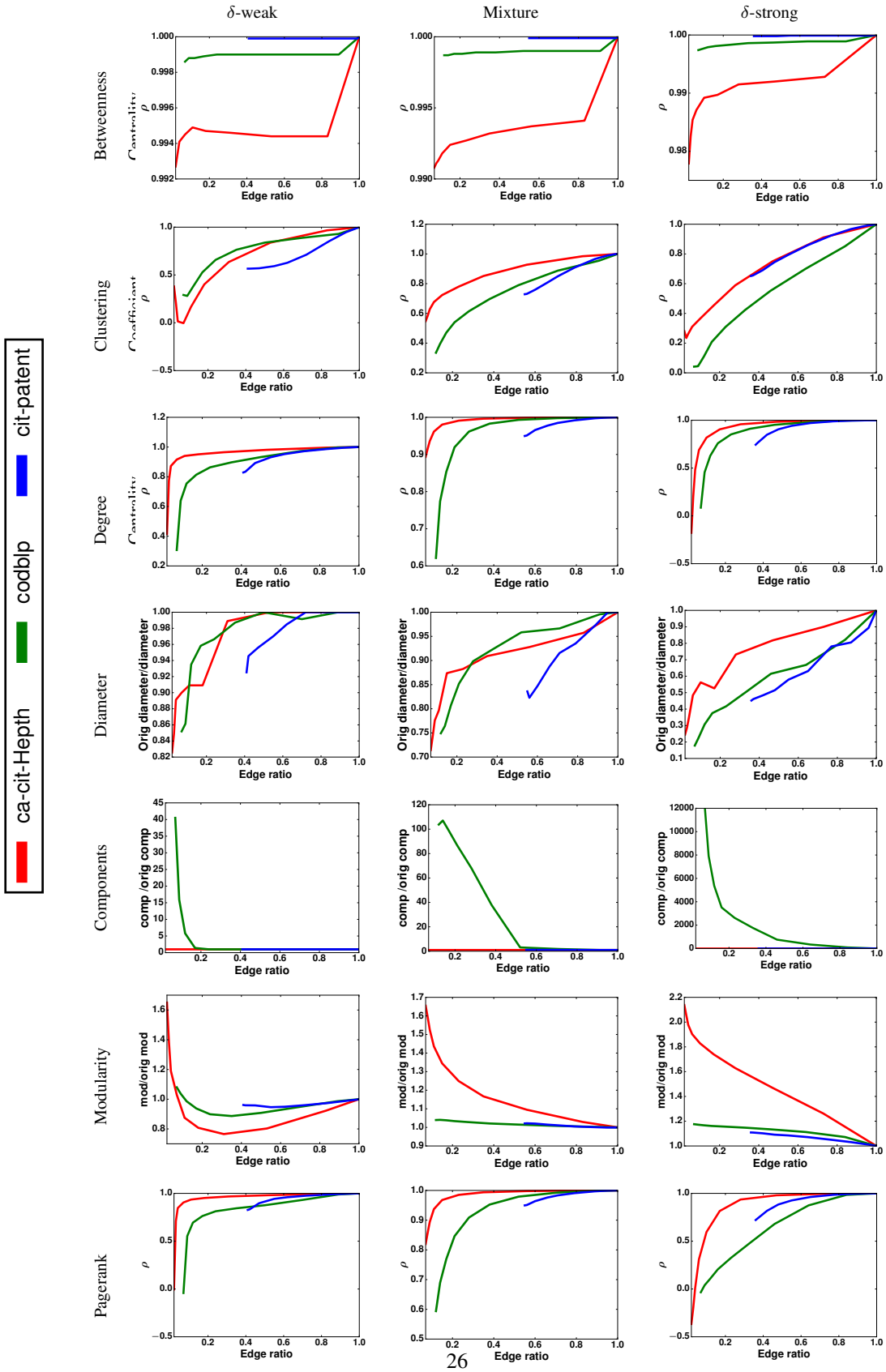


Figure 2.9: Citation Networks

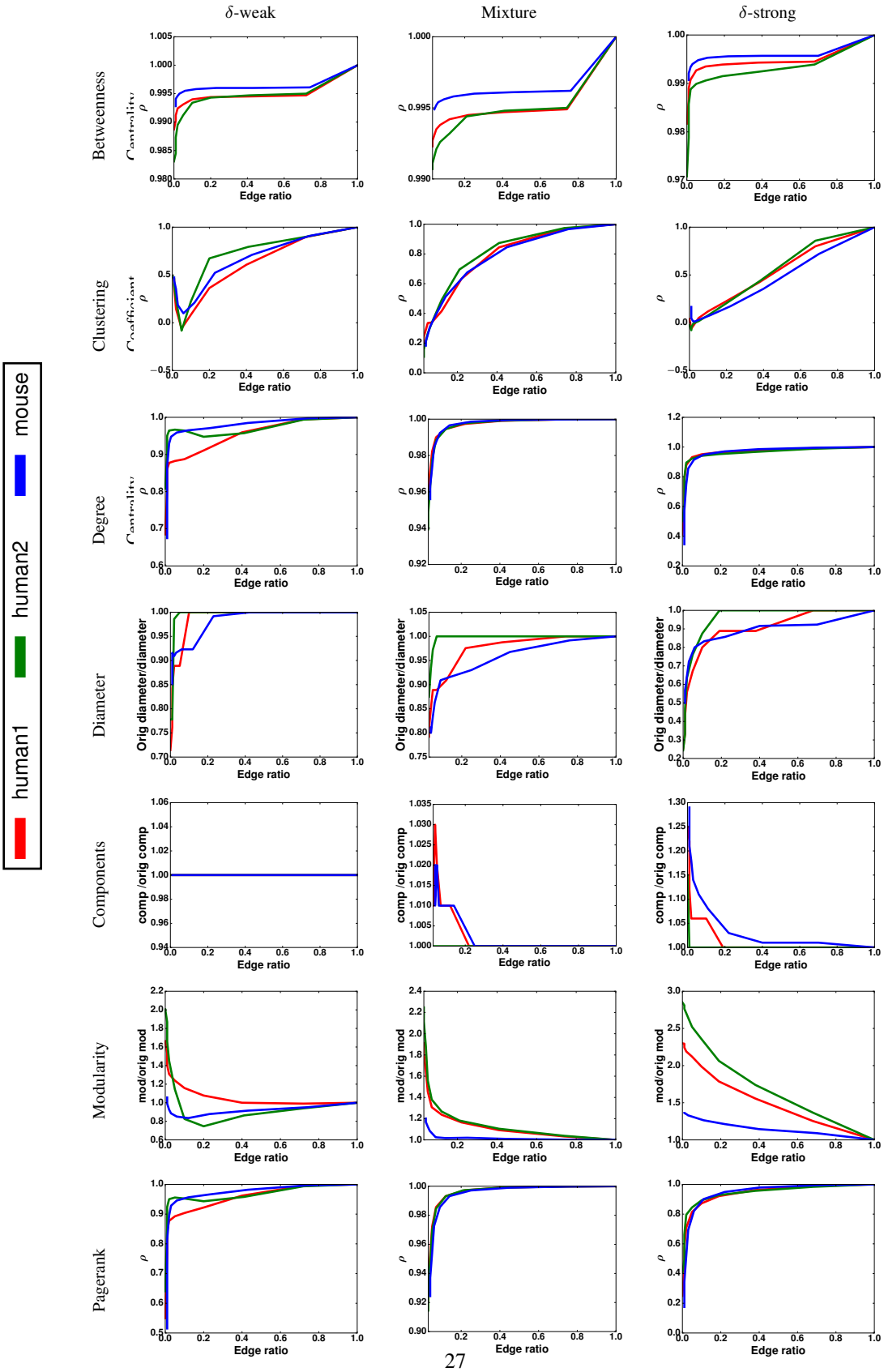


Figure 2.10: Biological Networks

Comparing with Local Degree In the introduction, we mentioned the Local Degree method (LD) [51] which favors the retention of edges participating in hubs (nodes with high degree). In order to compare our method to LD, we ran the single level algorithm for retaining weak edges, strong edges and a mixture of both on the Google+ graph (google-plus in Table 2.1). Same set of network properties discussed earlier in this section were used for comparison. For betweenness centrality, degree centrality, local clustering coefficient and PageRank, we plot the Spearman rank correlation against the edge ratio. Figure 2.11 shows the plots of δ-weak, δ-strong, mixed and local degree(LD) for each property. The results are similar for betweenness centrality, degree centrality and PageRank. However, for such properties as modularity and clustering coefficient, the algebraic distance performs better than LD especially when sparsification is aggressive. The δ-weak sparsification preserves the diameter slightly bettern than LD while the δ-strong method did not perform well on it and on the number of components. We note that the LD method was comprehensively studied on the Facebook networks only. Four Facebook networks in SN1 demonstrate similar performance with both methods. The Google+ network has exceptionally high clustering coefficient (0.52 vs. 0.23 in Facebook networks) and smaller diameter (6 vs. 8 in Facebook networks) which are more difficult to preserve if the method does not distinguish between local- and global-range connections.

2.7.1 Multi-Level Results

The purpose of the multilevel approach is to extend the general sparsification framework to enable highly controllable sparsification of bundles of edges at multiple coarse-grained resolutions. Similar to the single-level experiments, we group the networks into 4 groups. Tables 2.2, 2.3, 2.4, and 2.5 show the results of the multilevel algorithm for sets SN1, SN2, BIO and CIT, respectively. The 4 major column sections in the aforementioned tables consists of the graph name, the levels' configuration for which sparsifcation is tested, the number of edges at each setting, and the network properties that we study. The properties column consist of the following properties: a) CC - clustering coefficient b) D - Diameter of the graph, c) Q - Modularity of the graph, d) Γ - the number of components in the network, e) BC_ρ - Spearman rank correlation for betweenness centrality, f) PR_ρ - Spearman rank correlation of Pagerank, g) DC_ρ - Spearman rank correlation of degree centrality, and h) CC_ρ - Spearman rank correlation of the clustering coefficient. Correlation here represents the correlation between the original graph and the sparse graph. The "Level" column contains the sparsification settings at different levels, where G0 represents the original graph, G1 is the graph with sparsification only at the coarsest levels, G2 is the graph with sparsification only at the middle levels and G3 represents the graph with sparsification at the fine levels. In order to keep the results comparable, we keep the

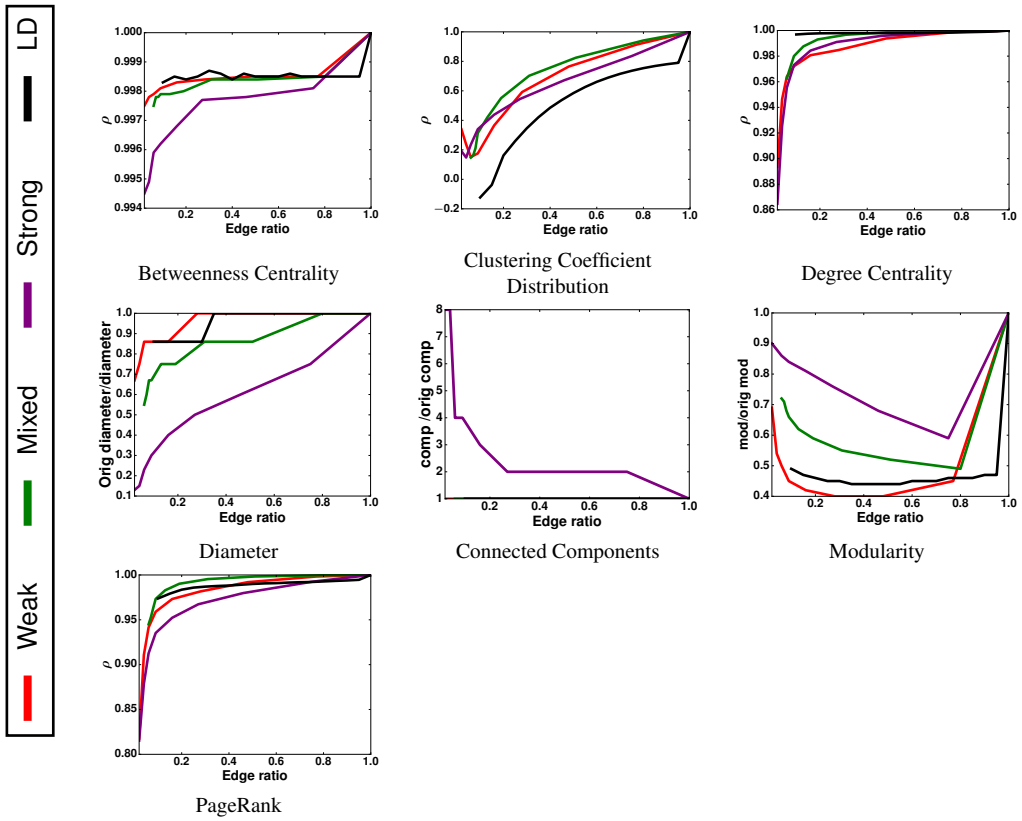


Figure 2.11: Comparison of LD and single-level algebraic distance methods.

sparsification ratio between 20% to 40%. The sparsification parameter is obtained by a binary search fitting algorithm. Note that we do not compare the sparse graphs (G1, G2, G3) to themselves but only with the fine graph G0. The parameter setting of a coarsening was similar to one described in [61] with interpolation order 1.

2.7.2 Running time

Figures 2.12(a-b) show the running time of both single- and multi-level algorithms for varying sparsification ratios. Each point in the plot represents the number of edges in the graph versus the runtime in seconds averaged over three runs. The coefficient of determination, R^2, shows how well the regression line fits the model. An R^2 of 0 indicates the line does not fit the data and an R^2 of 1 indicates the line perfectly fits the data. The results show that both algorithms scales linearly with the number of edges in the graph. As mentioned earlier, this is important as it defeats the purpose of sparsification if the algorithm is slow. The experiments were performed in a Linux environment on a multicore compute server with 64 Intel Xeon cores and 64 GB of memory.

Graph Name	Level	$\|E\|$	Properties							
			CC	D	Q	Γ	BC_ρ	PR_ρ	DC_ρ	CC_ρ
fb-indiana	G0	1.3M	0.21	8.0	0.45	1	1.0	1.0	1.0	1.0
	G1	361.4K	0.31	13.0	0.93	18	1.0	0.84	0.83	0.64
	G2	349.8K	0.14	10.0	0.34	8	0.99	0.89	0.89	0.57
	G3	402.7K	0.04	12.0	0.3	37	0.99	0.93	0.95	0.03
fb-texas84	G0	1.6M	0.2	7.0	0.38	1	1.0	1.0	1.0	1.0
	G1	374.7K	0.29	16.0	0.92	16	1.0	0.86	0.82	0.57
	G2	574.8K	0.13	9.0	0.31	5	0.99	0.94	0.94	0.66
	G3	521.9K	0.05	12.0	0.24	29	1.0	0.94	0.96	0.08
fb-uf	G0	1.5M	0.22	8.0	0.44	1	1.0	1.0	1.0	1.0
	G1	423.4K	0.24	12.0	0.61	3	0.99	0.88	0.88	0.58
	G2	425.1K	0.16	11.0	0.37	15	1.0	0.9	0.9	0.6
	G3	418.9K	0.05	11.0	0.27	57	1.0	0.92	0.95	-0.01
fb-penn94	G0	1.4M	0.22	8.0	0.48	1	1.0	1.0	1.0	1.0
	G1	351.7K	0.33	16.0	0.95	16	1.0	0.85	0.8	0.57
	G2	463.4K	0.16	11.0	0.38	23	1.0	0.89	0.9	0.6
	G3	365.6K	0.04	14.0	0.3	122	1.0	0.89	0.92	-0.09
livejournal	G0	34.7M	0.35	21.0	0.75	1	1.0	1.0	1.0	1.0
	G1	13.6M	0.47	72.0	1.0	1.7K	1.0	0.85	0.81	0.75
	G2	13.4M	0.39	42.0	0.8	7.1K	1.0	0.91	0.87	0.76
	G3	8.2M	0.02	30.0	0.72	316.9K	1.0	0.67	0.73	0.37
gplus	G0	12.2M	0.52	6.0	0.47	1	1.0	1.0	1.0	1.0
	G1	3.6M	0.54	14.0	0.89	15	1.0	0.91	0.9	0.59
	G2	3.0M	0.42	12.0	0.38	18	1.0	0.9	0.88	0.57
	G3	3.3M	0.15	19.0	0.26	905	1.0	0.79	0.88	-0.26

Table 2.2: Multiscale results for social networks 1 (SN1) graphs

2.7.3 Parallelization

Parallelization of the single-level algorithm does not require redesigning it. There are two computationally intensive parts of our method that gain from parallelization. One is the computation of the algebraic distance and the other the deletion of edges. Because of the implicitly parallel nature of the Jacobi over-relaxation, we are able to parallelize it by using OpenMP's shared data, multiple thread model. Since vector updates are independent, this method is highly efficient, creating speed gains of more than 50% with only 8 threads as seen in Figure 2.13. Figure 2.13 shows the benchmark results of parallelizing the algebraic distance computation where y-axis represents the average runtime averaged over 3 runs and x-axis represents the number of threads. We tested with number of threads ranging from 1 to 64 on 4 networks, namely, fb-uf, human-gene1, cit-patent, and catster. The experiments were performed in a Linux environment on a multicore compute server with 64 Intel Xeon cores and 64 GB of memory.

Graph Name	Level	$\|E\|$	Properties							
			CC	D	Q	Γ	BC_ρ	PR_ρ	DC_ρ	CC_ρ
flickr	G0	2.3M	0.09	9.0	0.67	83	1.0	1.0	1.0	1.0
	G1	921.0K	0.15	35.0	0.91	144	1.0	0.6	0.62	0.78
	G2	496.1K	0.12	18.0	0.84	134	1.0	0.71	0.73	0.8
	G3	634.9K	0.04	25.0	0.55	5.8K	1.0	0.64	0.77	0.74
buzznet	G0	2.8M	0.25	5.0	0.31	1	1.0	1.0	1.0	1.0
	G1	713.8K	0.26	11.0	0.5	1	1.0	0.82	0.91	0.6
	G2	919.3K	0.21	18.0	0.3	12	1.0	0.83	0.94	0.6
	G3	666.6K	0.1	16.0	0.29	239	1.0	0.75	0.89	0.28
foursquare	G0	3.2M	0.22	4.0	0.41	1	1.0	1.0	1.0	1.0
	G1	1.1M	0.17	36.0	0.94	18	1.0	0.74	0.88	0.87
	G2	765.9K	0.19	47.0	0.94	366	1.0	0.49	0.78	0.79
	G3	1.1M	0.04	14.0	0.57	10.0K	1.0	0.36	0.74	0.73
catster	G0	5.4M	0.41	10.0	0.39	281	1.0	1.0	1.0	1.0
	G1	1.3M	0.31	15.0	0.69	293	1.0	0.59	0.59	0.39
	G2	1.7M	0.26	11.0	0.37	360	1.0	0.86	0.87	0.63
	G3	1.5M	0.27	14.0	0.29	1.1K	1.0	0.76	0.84	0.4
blog-catalog	G0	2.1M	0.46	9.0	0.32	1	1.0	1.0	1.0	1.0
	G1	423.9K	0.41	14.0	0.67	1	1.0	0.81	0.87	0.47
	G2	570.8K	0.26	11.0	0.27	9	1.0	0.85	0.89	0.44
	G3	566.1K	0.18	12.0	0.23	391	1.0	0.7	0.83	0.29
livemocha	G0	2.2M	0.06	6.0	0.36	1	1.0	1.0	1.0	1.0
	G1	636.9K	0.04	12.0	0.45	3	1.0	0.89	0.91	0.48
	G2	869.1K	0.04	10.0	0.29	8	1.0	0.9	0.91	0.49
	G3	556.7K	0.02	11.0	0.29	1.4K	1.0	0.77	0.88	0.38

Table 2.3: Multiscale results for social networks 2(SN2) graphs

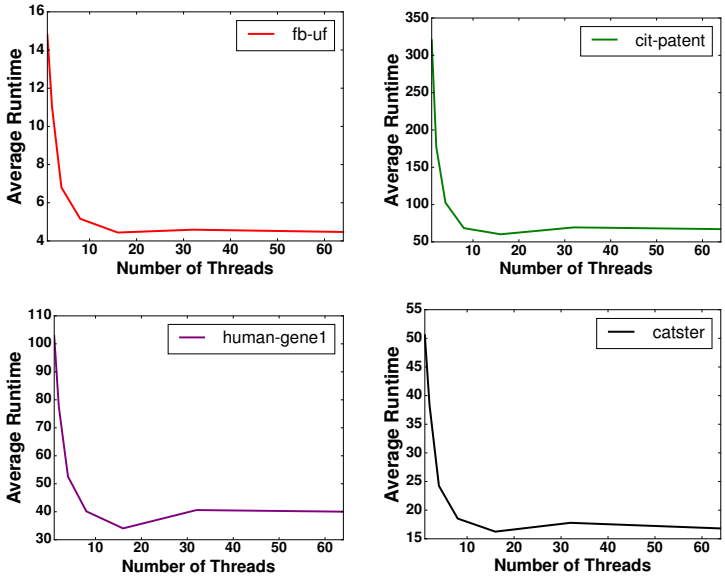


Figure 2.13: Running time in shared memory model.

Graph Name	Level	$\|E\|$	Properties							
			CC	D	Q	Γ	BC_ρ	PR_ρ	DC_ρ	CC_ρ
bio-human-gene1	G0	12.3M	0.63	8.0	0.38	17	1.0	1.0	1.0	1.0
	G1	2.8M	0.66	18.0	0.8	19	0.99	0.9	0.95	0.74
	G2	4.0M	0.33	9.0	0.39	22	0.99	0.91	0.91	0.71
	G3	3.9M	0.06	11.0	0.33	78	0.99	0.89	0.93	0.21
bio-human-gene2	G0	9.0M	0.66	7.0	0.31	2	1.0	1.0	1.0	1.0
	G1	2.4M	0.67	7.0	0.74	15	1.0	0.87	0.86	0.74
	G2	3.1M	0.64	26.0	0.52	14	0.98	0.87	0.88	0.74
	G3	2.5M	0.62	39.0	0.42	66	0.93	0.88	0.87	0.64
bio-mouse-gene	G0	14.5M	0.53	12.0	0.62	97	1.0	1.0	1.0	1.0
	G1	4.6M	0.6	21.0	0.89	105	0.99	0.94	0.95	0.72
	G2	4.1M	0.28	13.0	0.56	132	1.0	0.93	0.94	0.65
	G3	4.1M	0.06	14.0	0.52	400	1.0	0.91	0.95	0.08

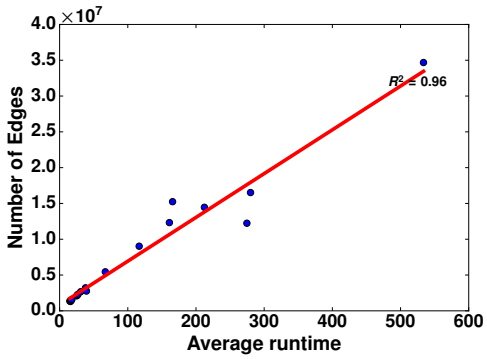
Table 2.4: Multiscale results for biological (BIO) networks

Graph Name	Level	$\|E\|$	Properties							
			CC	D	Q	Γ	BC_ρ	PR_ρ	DC_ρ	CC_ρ
ca-cit-Hepth	G0	2.4M	0.61	9.0	0.41	74	1.0	1.0	1.0	1.0
	G1	487.5K	0.7	18.0	0.93	86	0.99	0.84	0.84	0.75
	G2	875.3K	0.49	13.0	0.37	111	0.99	0.91	0.94	0.81
	G3	722.2K	0.21	16.0	0.25	279	0.99	0.83	0.95	-0.04
cit-patent	G0	16.5M	0.09	26.0	0.81	3.6K	1.0	1.0	1.0	1.0
	G1	5.8M	0.16	61.0	0.93	43.8K	1.0	0.79	0.78	0.73
	G2	3.4M	0.13	57.0	0.97	631.4K	1.0	0.58	0.64	0.59
	G3	5.0M	0.01	39.0	0.84	235.5K	1.0	0.68	0.74	0.54
codblp	G0	15.2M	0.82	23.0	0.84	1	1.0	1.0	1.0	1.0
	G1	5.3M	0.91	30.0	0.98	20.1K	1.0	0.55	0.78	0.68
	G2	3.4M	0.81	32.0	0.97	44.3K	1.0	0.27	0.63	0.62
	G3	4.7M	0.5	29.0	0.84	29.1K	1.0	0.51	0.82	0.38

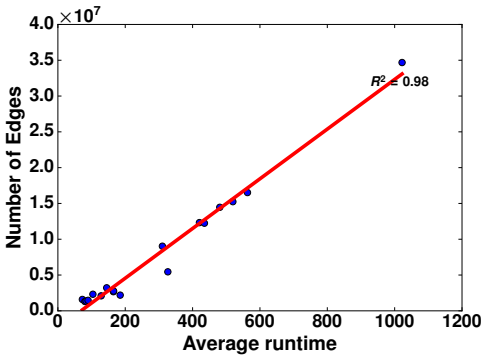
Table 2.5: Multiscale results for citation (CIT) networks

2.8 Conclusions

In this study we introduced single- and multi-level methods of network sparsification by algebraic distance. While many sparsification methods exist, most of them target certain properties without distinguishing short- and long-range connections that is the main goal of our method. We showed that by enabling different filtering capabilities, sparsification can be tuned to preserve either global or local structure or a combination of both. In addition to preserving a host of graph properties, we believe that the development of the multilevel sparsification framework can serve as a foundation for future work in that direction in which a variety of sparsification criteria (such as the algebraic distance) can be incorporated into it.



a) Single-level



b) Multi-level

Figure 2.12: Running time of sparsification.

Chapter 3

A Multilevel Algorithm for the Minimum 2-Sum Problem

3.1 Introduction

Linear ordering (or graph layout) problems are a class of computational optimization problems which deal with the permutation of the vertices of a graph in such a way that some objective is optimized subject to certain constraints. Many problems occurring in various disciplines such as VLSI design, numerical analysis, graph drawing, information retrieval and scheduling can be modeled as linear ordering problems (see [24, 35, 48, 75]). One of the most important applications is in the area of numerical analysis where the goal is to reorder the rows and columns of large sparse matrices such that the non-zero entries lie close to the diagonal thereby enabling efficient storage and improving the performance of the algorithms that depend on location of non-zeros such as Cholesky factorization [24].

Many versions of linear ordering problems are NP-hard and their decision versions are NP-complete which make them especially attractive for developing of new approximation algorithms. However, the practical importance also required development of fast and scalable heuristics that run in a linear (in the number of edges) time. For example, Juvan and Mohar applied spectral methods to the bandwidth, cutwidth and minimum p-sum problems in [42]. Other well known methods include various local search methods including randomized ones such as the simulated annealing [58], multilevel method for the minimum linear arrangement problem introduced by Koren et. al [44], multilevel method for the minimum wavefront reduction

[37], optimally oriented decomposition tree [5] and genetic hill-climbing [17]. Some methods are targeted towards improving the quality of the results while others focus on the execution time. Spectral methods were the first in which both questions have been successfully addressed. Their complexity mostly depends on the algorithm for computing the Fiedler vector of a graph Laplacian. However, recent studies have shown that multilevel methods outperform spectral methods both in execution time and quality of the solutions (see [65–67]).

In this chapter, we open with a discussion on different optimization objectives for linear ordering problems and the role of multilevel framework for solving these problems. Next we provide a multilevel framework algorithm that scales linearly and produces quality solution in practice. Our main contribution is in the introduction of a new asymmetric coarsening scheme for multilevel algorithms. By exploiting different asymmetric vertex aggregation algorithms in the coarsening process, we demonstrate that the current state of the art results for the minimum 2-sum problem are still far from being optimal, and substantially improve them for a certain class of networks.

3.1.1 Notation

Unless stated, all graphs in this chapter are simple and undirected. Given a graph $G = (V, E)$, the vertex set is denoted by V and the edge set is denoted by E. The edge between nodes i and j of the graph is represented as ij and w_{ij} denotes the nonnegative weight of the edge such that if $ij \notin E$ then $w_{ij} = 0$. The nonnegative volume of a node i is denoted as v_i. We denote the ordering on the nodes ψ as a bijection $\psi : V \to \{1, 2, ..., n\}$, where n is the number of nodes in G. Additionally, we define $\psi(G)$ as the set of all possible orderings of G. It then follows that a graph ordering can represented by mapping each vertex i to an integer position $\psi(i)$ on a horizontal line.

3.1.2 Minimum Linear Arrangement Problem

The minimum linear arrangement problem (MinLA) was originally formulated by Harper in 1964 [33]. It is applied in the field of VLSI where is used to the minimize the total length of wires used to connect logical gates [21]. In addition, it is applied to various fields of numerical analysis where large sparse matrix reordering is helpful and in graph drawing. Other applications can be found in [24, 35, 48, 75]. The MinLA problem is defined as follows. Given an edge weighted graph $G = (V, E)$ where $V = \{1, 2, ..n\}$ represents

the labeled vertices. The goal of the MinLA is to find an ordering on the nodes ψ such that

$$\psi = \sum_{ij \in E} w_{ij} |\psi(i) - \psi(j)| \tag{3.1}$$

is minimized.

In [66], the problem is further generalized to account the volumes of nodes. The minimization objective is redefined as

$$\sum_{ij \in E} w_{ij} |x_i - x_j|, \tag{3.2}$$

where for all $i \in V$ $x_i = \frac{v_i}{2} + \sum_{k,\psi(k)<\psi(i)} v_k$. The original problem is the case where all volumes are equal.

3.1.3 Minimum 2-sum Problem

The minimum 2-sum problem (M2sP) belongs to the same class of graph layout problems such as MinLA and finds application in many disciplines. The M2sP is also closely related to the problem of calculating the envelope size of a symmetric matrix or more precisely, to the amount of work needed in the Cholesky factorization. In addition, the M2sP may be motivated as a model used in VLSI design, where at the placement phase it is chosen to minimize the total squared wire length. The corresponding decision problem is also known to be NP-complete.

The goal of the M2sP is to find an ordering on the nodes ψ such that

$$\psi = \sum_{ij \in E} w_{ij} |\psi(i) - \psi(j)|^2$$

is minimized. Similar to MinLA, it is generalized for the vertex-volumed version with the minimization objective

$$\sum_{ij \in E} w_{ij} |x_i - x_j|^2, \tag{3.3}$$

where for all $i \in V$ $x_i = \frac{v_i}{2} + \sum_{k,\psi(k)<\psi(i)} v_k$.

3.1.4 Main Contribution

In this thesis our focus is primarily on the minimum 2-sum problem. However, the framework we developed can be extended to most of the other cost objectives. Our multilevel algorithm is based on the algo-

rithm developed in [66] which itself is based on the Algebraic Multigrid scheme (AMG) [10, 12, 13, 64, 81]. The AMG methods were first developed for solving linear systems of equations resulting partial differential equations (See Chapter 1). In our approach, we introduce and explore different asymmetric matching-based coarsening scheme, and compare their performance with AMG-based schemes. Particularly, we attempt to solve the classical stable matching problem for graph vertices resulting in a one-to-one matching between initial seed vertices and non-seed vertices. Finally, we provide the empirical results of our experiments, and demonstrate advantages of the proposed method.

3.2 The Algorithm

Given a graph G, in the multilevel framework we recursively create a hierarchy of graphs in increasing coarseness $G_0, G_1, ..., G_k$. The original graph is successively coarsened into smaller graphs until the graph is small enough to be solved. Typically a threshold is chosen to determine the size of the smallest graph G_k. However, it also depends on the available computational resources, because, at the coarsest level, the problem is solved exactly on the coarsest graph. This result is inherited successively by each graph during the uncoarsening. At each level of the uncoarsening process, small segments of the ordering are individually refined, keeping the runtime linear while improving the quality of the overall solution. In other words, we improve the global solution at the finest level by local operations at multiple levels of coarseness.

3.2.1 Coarsening

We explain the coarsening process using two graphs that are consecutive in the hierarchy. We define the fine-level graph as $G_f = (V_f, E_f)$ and the next coarser graph as $G_c = (V_c, E_c)$. The coarsening here is similar to the one developed in [66]. The main focus of our work is a combination of stable matching with AMG coarsening, but we describe the full algorithm for completeness of the thesis.

We begin by selecting an initial set of seed nodes $C \subset V_f$ that will serve as centers of future coarse nodes in V_c. We initialize $F = V_f$ and $C = \emptyset$, and then iteratively move vertices from F to C based on their relative strength of connection to C such that the nodes that are not strongly coupled to those that are already in C are given preference. At each step $F \cup C = V_f$ is preserved, and, at the end, the size of V_c is known, namely, $|V_c| = |C|$. This can be done in one pass through V_f. The future volume for each seed node is then

defined as

$$\vartheta_i = v_i + \sum_{j \in V} v_j . \frac{w_{ji}}{\sum\limits_{k \in V} w_{jk}} \tag{3.4}$$

Where $\frac{w_{ji}}{\sum\limits_{k \in V} w_{jk}}$ is the normalized weight of edge ji with respect to its neighbors. The future volume ϑ_i here is a measure of the possible capacity of the aggregate seed i. The C-node selection will accessed in the descending future volume order. We define a parameter μ such that nodes with future volume greater than μ times the average of all future volumes are selected as seed nodes. The selected seed nodes are removed from V and added to C, i.e., $F = V \setminus C$. Another parameter T is defined such that for each node $i \in F$, we add i to C if its weighted connection to C divided by its weighted degree is less than T. The algorithm for seed creation is defined in Algorithm 5.

Algorithm 5 Seed node selection

Input: Parameters μ, T, V
Output: Set of seed nodes C
$C \leftarrow \emptyset, F \leftarrow V$
Compute $\vartheta_i \; \forall i \in F$
$C \leftarrow$ nodes with $\vartheta_i > \mu \cdot (\bar{\vartheta})$
$F \leftarrow V \setminus C$
Recompute $\vartheta_i \; \forall i \in F$
Sort F in descending order of ϑ
for $i \in F$ **do**
    if $\frac{\sum_{j \in C} w_{ij}}{\sum_{j \in V} w_{ij}} \leq T$ **then**
        move i from F to C
    end if
end for
return C

After C is selected, nodes in $F = V \setminus C$ are distributed to their aggregates that form coarse nodes according to the restriction operator P which is a matrix of size $|V_f| \times |C|$ (see Equation 2.2) and $I_c(j)$ which returns an index of coarse node J that corresponds to $j \in C$. Then, the Galerkin coarsening creates a coarse graph Laplacian $L_c \leftarrow P^T L_f P$, where L_f is the Laplacian of G_f. See Algorithm 6 for the coarsening algorithm.

3.2.2 Stable Matching

The stable matching problem, referred to in some literature as the stable marriage problem, is the problem of finding a stable match between two disjoint set. The matching result is a one-to-one mapping of

Algorithm 6 Coarsening Algorithm (AMG)

Input: Fine Graph G_f
Output: Coarse graph G_c
function COARSEN(G_f)
    C $\leftarrow$ CREATESEEDS(G_f)
    for each edge ij in G_f **do**
        if $i \in F$ and $j \in N_i$ **then** ▷ N_i is the coarse neighborhood of node i
            $P_{ij} = w_{ij} / \sum_{k \in N_i} w_{ik}$
        else if $i \in C$ and $j = i$ **then**
            $P_{ij} = 1$
        else
            $P_{ij} = 0$
        end if
    end for
    Initialize $G_c(|C|)$
    for each coarse node p **do**
        for each coarse node q where $p \neq q$ **do**
            $w_{pq}^{coarse} = \sum_{k \neq l} P_{ki} w_{kl} P_{lj}$
        end for
        $v_p = \sum_j v_j P_{ji}$ ▷ Volume of coarse node p
    end for
    return G_c
end function

members in one set to another. Matching finds application in many real world problems. In particular, many coarsening schemes in multilevel algorithms employ different versions of weighted matching. Basically, matching algorithms help to answer the question on how to pair vertices during the process of coarsening to maximize the number of coarsened pairs.

In our framework, we provide an implementation of the classical algorithm developed by Gale and Shapley in [28]. The two disjoint sets consist of the set of seed nodes previously selected by the seed creation algorithm (Algorithm 3) and the set of non-seed nodes. Each seed vertex i maintains a preference list of non-seed vertices and likewise each non-seed vertex maintains a preference list of seed vertices. Edges are weighted by their *relative extended p-normed algebraic distances*, which is a new measure of the connectivity strength that we introduce in this work. The original extended p-normed algebraic distance was introduced in [20, 61], which is implemented in Algorithm 1. The corresponding strength of connectivity is defined as an inverse of

$$\rho_{ij} = \big(\sum_{r=1}^{R} |x_i^{(k,r)} - x_j^{(k,r)}|^p \big)^{\frac{1}{p}}, \tag{3.5}$$

where R is the number of randomly initialized test vectors, k is the number of iterations of Jacobi over-

relaxations applied on the homogeneous system of the corresponding graph Laplacian $Lx = 0$, and $x_i^{(k)}$ is the ith entry of the relaxed vector.

Although the original algebraic distance is symmetric (i.e., $\rho_{ij} = \rho_{ji}$), we can make it asymmetric by dividing each algebraic distance by the sum of the algebraic distances of the neighbors for both i, and j, namely, for all edges ij, we define

$$\varrho_{ij} = \frac{\rho_{ij}^{-1}}{\sum_j \rho_{ij}^{-1}}, \text{ and } \varrho_{ji} = \frac{\rho_{ij}^{-1}}{\sum_i \rho_{ij}^{-1}} \tag{3.6}$$

which becomes a new asymmetric measure for the strength of connectivity which will be reflected in the ordering of list of neighbors for the stable matching.

In stable matching, the stability is achieved where there are no two currently matched objects that prefers each other to their matched partners. Stability for unequal sets and weighted preference lists have been treated in previous studies (see [53, 59]). *We propose to use this in multilevel algorithms in the stages when one has to decide how to aggregate the variables. Most popular energy functions that describe the strength of connection between coarse and fine variables are designed in such a way that the decision process of connecting fine j to coarse i takes into account only one of the relative strengths of connection. Moreover, a similar reasoning is used to sparsify the restriction operator. We propose to bridge this gap by exploring stable matching-based multilevel schemes.* The result of the matching is a list of matched pairs of nodes. The nodes that are left unmatched are converted to seed nodes and this result is used to create the next coarse graph. See Algorithm 7 for the complete coarsening algorithm with stable matching.

3.2.3 Coarsest Level

At the coarsest level, we solve the problem exactly. For linear ordering, this involves generating all possible permutations and choosing the ordering with the minimum cost. The number of nodes here is chosen so that this does not become a bottleneck in the framework.

3.2.4 Uncoarsening

At each of the finer levels, the solution is initialized by inheriting the ordering from the coarser level and then placing the remaining F-nodes in such a way that the quadratic arrangement cost (3.3) is minimized. After initialization, the new ordering is further refined by multiple sweeps of Compatible and Gauss-Seidel relaxation, followed by node-by-node and window minimization (see [66]).

Algorithm 7 Coarsening Algorithm with stable matching

Input: Fine Graph G_f
Output: Coarse graph G_c
function COARSEN(G_f)
    $C \leftarrow$ CREATESEEDS(G_f)
    COMPUTEALGEBRAICDISTANCE(G_f)
    $matching \leftarrow$ GETSTABLEMATCHING(V_c, V_f)
    for each node i in V_f **do**
        if $matching[i]$ is unmatched **then**
            $C = \bigcup i$
        else
            $j \leftarrow matching[i]$
            $P_{ij} = 1$
        end if
    end for
    Initialize $G_c(|C|)$
    for each coarse node p **do**
        for each coarse node q where $p \neq q$ **do**
            $w_{pq}^{coarse} = \sum_{k \neq l} P_{ki} w_{kl} P_{lj}$
        end for
        $v_p = \sum_j v_j P_{ji}$ ▷ Volume of coarse node p
    end for
    return G_c
end function

3.2.4.1 Initialization

After solving the coarsest problem, we begin the process of uncoarsening at each level by first placing the seed nodes in order according to $y_j = x_I(j)$, where x_I is the coordinate of the coarse aggregate whose center of mass is j. Let V' be the nodes that have already been placed. Then initially, $V' = C$. Then for each fine node i, position i at y_i according to Equation 3.7 starting nodes whose relative connection to C is greater.

$$y_i = \frac{\sum_{j \in V'} y_j w_{ij}}{\sum_{j \in V'} w_{ij}} \tag{3.7}$$

Moving i to V' until $V' = V$. Then the ordering is made legal by converting y_i back to the x_i as defined in Equation 3.8.

$$x_i = \frac{v_i}{2} + \sum_{y_k < y_i} v_k \tag{3.8}$$

3.2.4.2 Relaxation

Initialization is followed by several relaxation sweeps. Compatible relaxation is first applied and then Gauss-Seidel (GS) relaxation. The relaxation process is similar to the initialization procedure. In Compatible, we only attempt to improve the position of the F-nodes using Equation 3.7 while $V' = V$. In GS, we repeat the process for all nodes in the graph.

3.2.4.3 Node-by-Node and Window Minimization

We further improve the arrangement by a node-by-node minimization. This involves moving each node k steps to the right and then to the left (with exceptions to the beginning and the end) and choosing the position with the minimum cost. This is essentially a fast process since the change is local and the cost at anytime can be computed using 2only the two adjacent nodes.

Additionally, we apply an advanced local minimization technique called window minimization used in [65, 66]. Window minimization tries to refine the arrangement cost of several nodes within a window by solving a system of equation for the window nodes. Let $\tilde{x}$ be the cost to the current arrangement, δ_i be the adjustment to $\tilde{x}$ and a window $W = \{i_1 = \psi^{-1}(p+1), ..., i_q = \psi^{-1}(p+q)\}$ of q sequential vertices, where p indicates the start of the window. The local cost minimization problem can thus be modeled as follows:

$$\text{minimize}\, \phi(W, \tilde{x}, \psi, \delta) = \sum_{i,j \in W} w_{ij}(\tilde{x}_i + \delta_i - \tilde{x}_j - \delta_j)^2 + \sum_{i \in W, j \notin W} w_{ij}(\tilde{x}_i + \delta_i - \tilde{x}_j)^2 \tag{3.9}$$

In order to take volume into account and restrict our objective $\{x_i + \delta_i\}_{i \in W}$, we add the following constraints:

$$\sum_{i \in W} (\tilde{x}_i + \delta_i)^m v_i = \sum_{i \in W} \tilde{x}_i^m v_i, \qquad m = 1, 2 \tag{3.10}$$

Using the Lagrange multipliers as applied in [66], the final formulation of the problem, yields the

system of equations given below:

$$\begin{cases} \sum_{j \in W} w_{ij}(\delta_i - \delta_j) + \delta_i \sum_{j \notin W} w_{ij} + \lambda_1 v_i + \lambda_2 v_i \tilde{x}_i = \sum_j w_{ij}(\tilde{x}_i - \tilde{x}_j) \quad i \in \{1...q\} \\ \sum_i \delta_i v_i = 0 \\ \sum_i \delta_i v_i \tilde{x}_i = 0 \end{cases} \tag{3.11}$$

For our implementation, we used the Eigen [30] solving the system. After solving the system, each vertex window is positioned at $y_i = \tilde{x} + \delta_i$ and then a legal ordering is enforced by applying Equation 3.8. The ordering is accepted if the new cost of the window improves the ordering. The full multilevel framework is presented in Algorithm 8

Algorithm 8 Multilevel framework for linear Ordering problems

```
1: Input: fine graph G_f = (V_f, E_f)
2: Output: Solved fine graph G_f
3: function ML(G_f)
4:     COMPUTEALGDIST(G_f)
5:     if |V_f| is small enough then
6:         SOLVEEXACTLY(G_f)                    ▷ Compute the exact solution
7:     else
8:         G_c ← COARSEN(G_f)                   ▷ Coarsening: coarse graph
9:         INITIALIZE(G_c, G_f)                 ▷ Coarse nodes become center of mass
10:        RELAXCOMPATIBLE(G_f)                 ▷ Compatible relaxation
11:        RELAXGS(G_f)                         ▷ Gauss-Seidel relaxation
12:        NODEMINIMIZE(G_f)                    ▷ Node-by-node minimization
13:        WINDOWMINIMIZE(G_f)                  ▷ Window minimization
14:    end if
15: end function
```

3.3 Results

Running stable matching at each level of the algorithm can be an expensive operation in the coarsening process since the nature of the unequal sets means the unmatched vertices are converted to seeds nodes so they can make it into the coarse graph. This can potentially slow down coarsening since the graph is coarsened less aggressively. In order to speed up the process we applied AMG coarsening at levels where difference in graphs produced by stable matching coarsening is less than 5%. This speeds up the coarsening especially for the large graphs.

We developed a C++ implementation of the multilevel algorithm and ran it on several input graphs on the palmetto super cluster, with Intel Xeon E5410 CPU, 8 cores, 64GB of memory. Since the algorithm is not parallel the major benefit from running on a cluster was being able to run jobs in batch. The algorithm was run on social networks and some finite element graphs. A description of the social network graphs can be found in Table 2.1 and the finite element graphs can be found in Table 3.2. In Table 3.1, we present the results of the multilevel algorithm. The terms AMG1, AMG2 and AMG10 represents pure AMG coarsening with interpolation order 1, 2, and 10 respectively. The term STABLE represents results where only stable matching is used in the coarsening scheme while STABLE+AMG represents results where stable-matching and AMG1 (interpolation order 1) are used during coarsening. Each column in the table shows ratio of the results compared to STABLE and STABLE+AMG. Column 1 contains the name of each graph. Column 2-4 compares each AMG scheme to the STABLE scheme. Column 5-7 compares AMG to STABLE+AMG and column 8 compares STABLE+AMG to STABLE.

Graph	AMG1/ STABLE	AMG2/ STABLE	AMG10/ STABLE	AMG1/ STABLE+AMG	AMG2/ STABLE+AMG	AMG10/ STABLE+AMG	STABLE+AMG/ STABLE
144	1	0.99	0.99	1.00	1.00	0.99	0.99
598a	0.97	0.97	0.97	0.98	0.98	0.98	0.99
auto	0.80	0.80	0.80	0.97	0.97	0.97	0.82
bcsstk30	0.81	0.77	0.76	0.92	0.87	0.87	0.88
blogcatalog	1.60	1.60	1.60	1.22	1.22	1.23	1.31
fb-uf	1.03	1.08	1.09	1.01	1.06	1.07	1.02
fe_ocean	0.94	0.87	0.88	0.99	0.91	0.92	0.95
finan	1.79	1.16	1.03	1.78	1.15	1.02	1.02
foursquare	1.25	1.34	1.37	1.20	1.28	1.31	1.05

Table 3.1: Results for multilevel minimum 2-sum solver for AMG, STABLE and STABLE+AMG

Table 3.2: Benchmark graphs (finite element structures)

Graph	$\|V\|$	$\|E\|$
144	144649	1074393
598a	110971	741943
bcsstk30	44609	985046
fe_ocean	143437	409593
finan	74752	261120
auto	448695	3314611

Our results show that by using stable matching in the coarsening scheme, we are able to greatly improve the quality of results for the multilevel M2sP solver. For instance we see a 60% improvement for graphs like blogcatalog, and finan. In general, social networks show some improvements in the results. However, finite element graphs like 144 and 598a did not show any significant improvements. We postulate that the improvement in social networks is because of their star-like (heavy tail) structure although more study is still needed to verify this.

3.4 Conclusion

In this section, we experimented with matching based asymmetric coarsening schemes. We developed the stable matching algorithm for the coarsening scheme of a multilevel minimum 2-sum solver. In applying stable matching in the coarsening scheme we explored the asymmetric property of this kind of matching and how it can improve our results. We demonstrated that stable matching can be used to improve the M2sP results for some social networks but shows no significant difference for finite element graphs. Future work is still needed to fully explore the role of this method. However, our study provide a good starting point for more research because, to the best of our knowledge, no algorithm has been able to improve the results in [66] in a reasonable computational time. The most beneficial class of networks contains instances with the heavy-tail degree distribution.

Bibliography

[1] Nesreen K. Ahmed, Jennifer Neville, and Ramana Kompella. Network sampling: From static to streaming graphs. ACM Trans. Knowl. Discov. Data, 8(2):7:1–7:56, jun 2013.

[2] David Auber. Tulipa huge graph visualization framework. In Graph Drawing Software, pages 105–126. Springer, 2004.

[3] David A Bader, Shiva Kintali, Kamesh Madduri, and Milena Mihail. Approximating betweenness centrality. In Algorithms and Models for the Web-Graph, pages 124–137. Springer, 2007.

[4] N.S. Bakhvalov. On the convergence of a relaxation method under natural constraints on an elliptic operator. Zhurnal vychislitel'noj matemaki i matematicheskoj fiziki, 6:861–883, 1966.

[5] R. Bar-Yehuda, G. Even, J. Feldman, and J. Naor. Computing an optimal orientation of a balanced decomposition tree for linear arrangement problems. Journal of Graph Algorithms and Applications, 5(4):1–27, 2001.

[6] Mathieu Bastian, Sebastien Heymann, Mathieu Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. ICWSM, 8:361–362, 2009.

[7] P. M. Binder. Frustration in complexity. Science, 322:323, 2008.

[8] A. Brandt. Multi-level adaptive technique (MLAT) for fast numerical solutions to boundary value problems. In H. Cabannes and R. Temam, editors, Lecture Notes in Physics 18, pages 82–89, pub-SV:adr, 1973. Proc. 3rd Int. Conf. Numerical Methods in Fluid Mechanics, pub-SV.

[9] A. Brandt. Multigrid techniques: 1984 guide, with applications to fluid dynamics. Gmd studien nr. 85, GMD, GMD-AIW, Postfach 1240, D-5205, St. Augustin 1, Germany, 1984. 191 pages.

[10] A. Brandt. Algebraic multigrid theory: The symmetric case. Jornal of Appl. Math. Comp., 19:23–56, 1986. Preliminary proceedings of the International Multigrid Conference, April 6–8, 1983, Copper Mountain, CO.

[11] A. Brandt. Multiscale Scientific Computation: Review 2001. In Multiscale and Multiresolution Methods, volume 20 of LNCSE, pages 3–95. Springer, 2002.

[12] A. Brandt, S. McCormick, and J. Ruge. Algebraic multigrid (AMG) for automatic multigrid solution with application to geodetic computations. Technical report, Institute for Computational Studies, Fort Collins, CO, POB 1852, 1982.

[13] A. Brandt, S. McCormick, and J. Ruge. Algebraic multigrid (AMG) for sparse matrix equations. In D. J. Evans, editor, Sparsity and its Applications, pages 257–284, 1984.

[14] A. Brandt and D. Ron. Chapter 1 : Multigrid solvers and multilevel optimization strategies. In J. Cong and J. R. Shinnerl, editors, Multilevel Optimization and VLSICAD. Kluwer, 2003.

[15] A. Brandt, D. Ron, and D. Amit. Multi-level approaches to discrete-state and stochastic problems. In W. Hackbush and U. Trottenberg, editors, Multigrid Methods II, pages 66–99. Springer-Verlag, 1986.

[16] Achi Brandt, James J. Brannick, Karsten Kahl, and Irene Livshits. Bootstrap AMG. SIAM J. Scientific Computing, 33(2):612–632, 2011.

[17] T. N. Bui and B. R. Moon. Genetic algorithm and graph partitioning. IEEE Trans. Comput., 45(7):841–855, 1996.

[18] Pavel Chebotarev and Elena Shamis. On proximity measures for graph vertices. arXiv preprint math/0602073, 2006.

[19] Jie Chen and Ilya Safro. Algebraic distance on graphs. SIAM Journal on Scientific Computing, 33(6):3468–3490, 2011.

[20] Jie Chen and Ilya Safro. Algebraic distance on graphs. SIAM Journal on Scientific Computing, (in press), 2011.

[21] C.-K. Cheng. Linear placement algorithm and applications to VLSI design. Netw., 17(4):439–464, 1987.

[22] Fan Chung, Wenbo Zhao, and Mark Kempton. Ranking and sparsifying a connection graph. Internet Mathematics, 10(1-2):87–115, 2014.

[23] Wouter De Nooy, Andrej Mrvar, and Vladimir Batagelj. Exploratory social network analysis with Pajek, volume 27. Cambridge University Press, 2011.

[24] J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. ACM Comput. Surv., 34(3):313–356, 2002.

[25] Hristo N Djidjev. A scalable multilevel algorithm for graph clustering and community structure detection. In Algorithms and models for the web-graph, pages 117–128. Springer, 2006.

[26] R.P. Fedorenko. A relaxation method for solving elliptic difference equations. Zhurnal vychislitel'noj matemaki i matematicheskoj fiziki, 1:922–927, 1961.

[27] R.P. Fedorenko. The speed of convergence of one iteration process. Zhurnal vychislitel'noj matemaki i matematicheskoj fiziki, 4:559–563, 1964.

[28] David Gale and Lloyd S Shapley. College admissions and the stability of marriage. The American Mathematical Monthly, 69(1):9–15, 1962.

[29] Leo A Goodman. Snowball sampling. The annals of mathematical statistics, pages 148–170, 1961.

[30] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[31] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In Proceedings of the 7th Python in Science Conference (SciPy2008), pages 11–15, Pasadena, CA USA, August 2008.

[32] M. S. Handcock and K. J. Gile. On the Concept of Snowball Sampling. ArXiv e-prints, 2011.

[33] L. H. Harper. Optimal assignments of numbers to vertices. Journal of the Society for Industrial and Applied Mathematics, 12(1):131–135, March 1964.

[34] B. Hendrickson and R. Leland. A multi-level algorithm for partitioning graphs. In Proceedings Supercomputing '95, page 28 (CD). ACM Press, 1995.

[35] Steven Bradish Horton. The Optimal Linear Arrangement Problem: Algorithms And Approximation. PhD thesis, Georgia Institute of Technology, May 1997.

[36] Pili Hu and Wing Cheong Lau. A survey and taxonomy of graph sampling. CoRR, abs/1308.5865, 2013.

[37] Y. F. Hu and J. A. Scott. A multilevel algorithm for wavefront reduction. SIAM J. Sci. Comput., 23(4):1352–1375, 2001.

[38] Yifan Hu. Efficient, high-quality force-directed graph drawing. Mathematica Journal, 10(1):37–71, 2005.

[39] S. Itzkovitz, R. Levitt, N. Kashtan, R. Milo, M. Itzkovitz, and U. Alon. Coarse-graining and self-dissimilarity of complex networks. Physical Review E, 71(1):016127, 2005.

[40] Emmanuel John and Ilya Safro. Network sparsification by algebraic distance. `https://github.com/emmanuj/ml-sparsifier`, 2015 – 2016.

[41] Emmanuel John and Ilya Safro. Single- and multi-level network sparsification by algebraic distance. CoRR, abs/1601.05527, 2016.

[42] Martin Juvan and Bojan Mohar. Optimal linear labelings and eigenvalues of graphs. Discrete Appl. Math., 36(2):153–168, 1992.

[43] Risi Imre Kondor and John Lafferty. Diffusion kernels on graphs and other discrete structures. In Proceedings of the 19th international conference on machine learning, pages 315–322, 2002.

[44] Y. Koren and D. Harel. Multi-scale algorithm for the linear arrangement problem. Proceedings of 28th Inter. Workshop on Graph-Theoretic Concepts, 2002.

[45] Jérôme Kunegis. Konect: The koblenz network collection. In Proceedings of the 22Nd International Conference on World Wide Web, WWW '13 Companion, pages 1343–1350, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.

[46] Maciej Kurant, Minas Gjoka, Yan Wang, Zack W. Almquist, Carter T. Butts, and Athina Markopoulou. Coarse-grained topology estimation via graph sampling. CoRR, abs/1105.5488, 2011.

[47] Dan Kushnir, Meirav Galun, and Achi Brandt. Fast multiscale clustering and manifold identification. Pattern Recognition, 39(10):1876–1891, 2006.

[48] Y. Lai and K. Williams. A survey of solved problems and applications on bandwidth, edgesum, and profile of graphs. J. Graph Theory, 31:75–94, 1999.

[49] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[50] Jure Leskovec and Rok Sosič. SNAP: A general purpose network analysis and graph mining library in C++. `http://snap.stanford.edu/snap`, June 2014.

[51] Gerd Lindner, Christian L. Staudt, Michael Hamann, Henning Meyerhenke, and Dorothea Wagner. Structure-preserving sparsification of social networks. In Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015, ASONAM '15, pages 448–454, New York, NY, USA, 2015. ACM.

[52] Oren E Livne and Achi Brandt. Lean algebraic multigrid (lamg): Fast graph laplacian linear solver. SIAM Journal on Scientific Computing, 34(4):B499–B522, 2012.

[53] DG McVitie and Leslie B Wilson. Stable marriage assignment for unequal sets. BIT Numerical Mathematics, 10(3):295–309, 1970.

[54] Enys Mones, Lilla Vicsek, and Tamas Vicsek. Hierarchy measure for complex networks. PLoS ONE, 7(3):e33799, 03 2012.

[55] Stephen G Nash. A multigrid approach to discretized optimization problems. Optimization Methods and Software, 14(1-2):99–116, 2000.

[56] M. E. J. Newman. Networks, An Introduction. Oxford University Press, 2010.

[57] Andreas Noack and Randolf Rotta. Multi-level algorithms for modularity clustering. CoRR, abs/0812.4073, 2008.

[58] J. Petit. Combining spectral sequencing and parallel simulated annealing for the minla problem. Parallel Processing Letters, 13(1):77–91, 2003.

[59] Maria Silvia Pini, Francesca Rossi, K Brent Venable, and Toby Walsh. Stability, optimality and manipulation in matching problems with weighted preferences. Algorithms, 6(4):782–804, 2013.

[60] Talayeh Razzaghi and Ilya Safro. Scalable multilevel support vector machines. Procedia Computer Science, 51:2683–2687, 2015.

[61] Dorit Ron, Ilya Safro, and Achi Brandt. Relaxation-based coarsening and multiscale graph organization. Multiscale Modeling & Simulation, 9(1):407–423, 2011.

[62] Ryan A. Rossi and Nesreen K. Ahmed. bio-human-gene1 - biological networks, 2013.

[63] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.

[64] J. Ruge and K. Stüben. Algebraic Multigrid, pages 73–130. SIAM, 1987.

[65] I. Safro, D. Ron, and A. Brandt. Graph minimum linear arrangement by multilevel weighted edge contractions. Journal of Algorithms, 60(1):24–41, 2006.

[66] I. Safro, D. Ron, and A. Brandt. Multilevel algorithm for the minimum 2-sum problem. Journal of Graph Algorithms and Applications, 10(2):237–258, 2006.

[67] I. Safro, D. Ron, and A. Brandt. Multilevel algorithms for linear ordering problems. ACM Journal of Experimental Algorithmics, 13:1.4–1.20, 2008.

[68] Ilya Safro, Dorit Ron, and Achi Brandt. Multilevel algorithms for linear ordering problems. Journal of Experimental Algorithmics (JEA), 13:4, 2009.

[69] Ilya Safro, Peter Sanders, and Christian Schulz. Advanced coarsening schemes for graph partitioning. Journal of Experimental Algorithmics (JEA), 19:2–2, 2015.

[70] Ilya Safro and Boris Temkin. Multiscale approach for the network compression-friendly ordering. J. Discrete Algorithms, 9(2):190–202, 2011.

[71] Tanwistha Saha, Huzefa Rangwala, and Carlotta Domeniconi. Sparsification and sampling of networks for collective classification. In Social Computing, Behavioral-Cultural Modeling and Prediction, pages 293–302. Springer, 2013.

[72] Sophia Sakellaridi, Haw-ren Fang, and Yousef Saad. Graph-based multilevel dimensionality reduction with applications to eigenfaces and latent semantic indexing. In Machine Learning and Applications, 2008. ICMLA'08. Seventh International Conference on, pages 194–200. IEEE, 2008.

[73] Matthew J Salganik and Douglas D Heckathorn. Sampling and estimation in hidden populations using respondent-driven sampling. Sociological methodology, 34(1):193–240, 2004.

[74] Venu Satuluri, Srinivasan Parthasarathy, and Yiye Ruan. Local graph sparsification for scalable clustering. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, pages 721–732. ACM, 2011.

[75] Farhad Shahrokhi, Ondrej Sýkora, László A. Székely, and Imrich Vrto. On bipartite drawings and the linear arrangement problem. SIAM Journal on Computing, 30(6):1773–1789, 2001.

[76] Eitan Sharon, Meirav Galun, Dahlia Sharon, Ronen Basri, and Achi Brandt. Hierarchy and adaptivity in segmenting visual scenes. Nature, 442(7104):810–813, 2006.

[77] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. SIAM Journal on Computing, 40(6):1913–1926, 2011.

[78] Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, pages 81–90. ACM, 2004.

[79] Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. SIAM Journal on Computing, 40(4):981–1025, 2011.

[80] Christian Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. Networkit: An interactive tool suite for high-performance network analysis. CoRR, abs/1403.3005, 2014.

[81] Klaus Stüben. An introduction to algebraic multigrid. In U. Trottenberg, C. W. Oosterlee, and A. Schüller, editors, Multigrid, pages 413–532. Academic Press, 2000. Appendix A.

[82] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger. Sampling techniques for large, dynamic graphs. In INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings, pages 1–6, April 2006.

[83] Arthur D Szlam, Mauro Maggioni, and Ronald R Coifman. Regularization on graphs with function-adapted diffusion processes. The Journal of Machine Learning Research, 9:1711–1739, 2008.

[84] Todd L. Veldhuizen. Dynamic multilevel graph visualization. CoRR, abs/0712.1549, 2007.

[85] C. Walshaw. A multilevel approach to the travelling salesman problem. Tech. Rep. 00/IM/63, Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK, August 2000.

[86] C. Walshaw. Multilevel refinement for combinatorial optimisation problems. Annals Oper. Res., 131:325–372, 2004.

[87] Chris Walshaw. A multilevel algorithm for force-directed graph drawing. In Proceedings of the 8th International Symposium on Graph Drawing, GD '00, pages 171–182, London, UK, UK, 2001. Springer-Verlag.

[88] Tianyi Wang, Yang Chen, Zengbin Zhang, Tianyin Xu, Long Jin, Pan Hui, Beixing Deng, and Xing Li. Understanding graph sampling algorithms for social network analysis. In Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on, pages 123–128. IEEE, 2011.

[89] D.H. Wolpert and W. Macready. Using self-dissimilarity to quantify complexity. Complexity, 12(3):77–85, 2007.